An introduction to GRASP

Mauricio G. C. Resende
AT&T Labs – Research, Florham Park, USA

XXXIX Simpósio Brasileiro de Pesquisa Operacional
August 28–31, 2007, Fortaleza, Ceará, Brazil

This book is dedicated to

Renata Machado Aiex (in Memorium)

Contents

Preface

This primer on greedy randomized adaptive search procedures, or simply GRASP, is intended to serve a complementary material for a four-hour short course on GRASP given at the 39th annual meeting of the Brazilian Operational Research Society (*XXXIX Simpósio Brasileiro de Pesquisa Operacional*) in Fortaleza, Ceará, Brazil from August 28 to 31, 2007.

The book is made up of four parts in 13 chapters. Each chapter has been adapted from a previously published paper. In the first part, the chapters cover a general introduction to GRASP, GRASP with path-relinking and evolutionary path-relinking, and GRASP with perturbations. In the second part, we cover parallel GRASP. Chapters there focus on an introduction to parallel GRASP, TTT plots and its use in the analysis of algorithms, parallel GRASP with path-relinking, probability distribution of running time in GRASP, and an application of parallel GRASP to job shop scheduling. The third part of the book illustrates efficient implementations of GRASP. We focus on two location problems and show, in three chapters, how GRASP can be used to find high-quality solutions to these problems. Finally, the last part of the book is a single chapter with an annotated bibliography.

I would like to acknowledge and thank by long time collaborators who coauthored these chapters with me. They are Renata Aiex (Chapters 5, 6, 8, and 9), Diogo Andrade (Chapter 4), Silvio Binato (Chapter 9), Suzana Canuto (Chapter 3), Paola Festa (Chapter 13), Celso Ribeiro (Chapters 1, 2, 3, 5, 6, and 7), and Renato Werneck (10, 11, and 12).

Mauricio G. C. Resende
Florham Park
August 2007

I GRASP and Path-Relinking

1 GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURES

Mauricio G. C. Resende[1] and Celso C. Ribeiro[2]

[1]Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA
mgcr@research.att.com

[2]Department of Computer Science
Catholic University of Rio de Janeiro
Rio de Janeiro, RJ 22453-900 Brazil
celso@inf.puc-rio.br

Abstract: GRASP is a multi-start metaheuristic for combinatorial problems, in which each iteration consists basically of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is investigated until a local minimum is found during the local search phase. The best overall solution is kept as the result. In this chapter, we first describe the basic components of GRASP. Successful implementation techniques and parameter tuning strategies are discussed and illustrated by numerical results obtained for different applications. Enhanced or alternative solution construction mechanisms and techniques to speed up the search are also described: Reactive GRASP, cost perturbations, bias functions, memory and learning, local search on partially constructed solutions, hashing, and filtering. We also discuss in detail implementation strategies of memory-based intensification and post-optimization techniques using path-relinking. Hybridizations with other metaheuristics, parallelization strategies, and applications are also reviewed.
Keywords: GRASP, metaheuristic, local search, greedy randomized search.

1.1 INTRODUCTION

We consider in this chapter a combinatorial optimization problem, defined by a finite ground set $E = \{1, \ldots, n\}$, a set of feasible solutions $F \subseteq 2^E$, and an objective function $f : 2^E \rightarrow \mathbb{R}$. In the minimization version, we search an optimal solution $S^* \in F$ such that $f(S^*) \leq f(S)$, $\forall S \in F$. The ground set E, the cost function f, and the set of

feasible solutions F are defined for each specific problem. For instance, in the case of the traveling salesman problem, the ground set E is that of all edges connecting the cities to be visited, $f(S)$ is the sum of the costs of all edges $e \in S$, and F is formed by all egde subsets that determine a Hamiltonian cycle.

The GRASP (Greedy Randomized Adaptive Search Procedure) metaheuristic (Feo and Resende, 1989; 1995) is a multi-start or iterative process, in which each iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is investigated until a local minimum is found during the local search phase. The best overall solution is kept as the result. An extensive survey of the literature is given in Festa and Resende (2002). The pseudo-code in Figure 1.1 illustrates the main blocks of a GRASP procedure for minimization, in which `Max_Iterations` iterations are performed and `Seed` is used as the initial seed for the pseudorandom number generator.

```
procedure GRASP(Max_Iterations,Seed)
1    Read_Input();
2    for k = 1,...,Max_Iterations do
3        Solution ← Greedy_Randomized_Construction(Seed);
4        Solution ← Local_Search(Solution);
5        Update_Solution(Solution,Best_Solution);
6    end;
7    return Best_Solution;
end GRASP.
```

Figure 1.1 Pseudo-code of the GRASP metaheuristic.

Figure 1.2 illustrates the construction phase with its pseudo-code. At each iteration of this phase, let the set of candidate elements be formed by all elements that can be incorporated to the partial solution under construction without destroying feasibility. The selection of the next element for incorporation is determined by the evaluation of all candidate elements according to a greedy evaluation function. This greedy function usually represents the incremental increase in the cost function due to the incorporation of this element into the solution under construction. The evaluation of the elements by this function leads to the creation of a restricted candidate list (RCL) formed by the best elements, i.e. those whose incorporation to the current partial solution results in the smallest incremental costs (this is the greedy aspect of the algorithm). The element to be incorporated into the partial solution is randomly selected from those in the RCL (this is the probabilistic aspect of the heuristic). Once the selected element is incorporated to the partial solution, the candidate list is updated and the incremental costs are reevaluated (this is the adaptive aspect of the heuristic). This strategy is similar to the semi-greedy heuristic proposed by Hart and Shogan (1987), which is also a multi-start approach based on greedy randomized constructions, but without local search.

The solutions generated by a greedy randomized construction are not necessarily optimal, even with respect to simple neighborhoods. The local search phase usu-

```
procedure Greedy_Randomized_Construction(Seed)
1    Solution ← ∅;
2    Evaluate the incremental costs of the candidate elements;
3    while Solution is not a complete solution do
4         Build the restricted candidate list (RCL);
5         Select an element s from the RCL at random;
6         Solution ← Solution ∪ {s};
7         Reevaluate the incremental costs;
8    end;
9    return Solution;
end Greedy_Randomized_Construction.
```

Figure 1.2 Pseudo-code of the construction phase.

ally improves the constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution in the neighborhood of the current solution. It terminates when no better solution is found in the neighborhood. The pseudo-code of a basic local search algorithm starting from the solution `Solution` constructed in the first phase and using a neighborhood N is given in Figure 1.2.

```
procedure Local_Search(Solution)
1    while Solution is not locally optimal do
2         Find s' ∈ N(Solution) with f(s') < f(Solution);
3         Solution ← s';
4    end;
5    return Solution;
end Local_Search.
```

Figure 1.3 Pseudo-code of the local search phase.

The effectiveness of a local search procedure depends on several aspects, such as the neighborhood structure, the neighborhood search technique, the fast evaluation of the cost function of the neighbors, and the starting solution itself. The construction phase plays a very important role with respect to this last aspect, building high-quality starting solutions for the local search. Simple neighborhoods are usually used. The neighborhood search may be implemented using either a *best-improving* or a *first-improving* strategy. In the case of the best-improving strategy, all neighbors are investigated and the current solution is replaced by the best neighbor. In the case of a first-improving strategy, the current solution moves to the first neighbor whose cost function value is smaller than that of the current solution. In practice, we observed on many applications that quite often both strategies lead to the same final solution, but in smaller computation times when the first-improving strategy is used. We also

observed that premature convergence to a non-global local minimum is more likely to occur with a best-improving strategy.

1.2 CONSTRUCTION OF THE RESTRICTED CANDIDATE LIST

An especially appealing characteristic of GRASP is the ease with which it can be implemented. Few parameters need to be set and tuned. Therefore, development can focus on implementing efficient data structures to assure quick iterations. GRASP has two main parameters: one related to the stopping criterion and another to the quality of the elements in the restricted candidate list.

The stopping criterion used in the pseudo-code described in Figure 1.1 is determined by the number `Max_Iterations` of iterations. Although the probability of finding a new solution improving the currently best decreases with the number of iterations, the quality of the best solution found may only improve with the latter. Since the computation time does not vary much from iteration to iteration, the total computation time is predictable and increases linearly with the number of iterations. Consequently, the larger the number of iterations, the larger will be the computation time and the better will be the solution found.

For the construction of the RCL used in the first phase we consider, without loss of generality, a minimization problem as the one formulated in Section 1.1. We denote by $c(e)$ the incremental cost associated with the incorporation of element $e \in E$ into the solution under construction. At any GRASP iteration, let c^{min} and c^{max} be, respectively, the smallest and the largest incremental costs.

The restricted candidate list RCL is made up of elements $e \in E$ with the best (i.e., the smallest) incremental costs $c(e)$. This list can be limited either by the number of elements (cardinality-based) or by their quality (value-based). In the first case, it is made up of the p elements with the best incremental costs, where p is a parameter. In this chapter, the RCL is associated with a threshold parameter $\alpha \in [0,1]$. The restricted candidate list is formed by all "feasible" elements $e \in E$ which can be inserted into the partial solution under construction without destroying feasibility and whose quality is superior to the threshold value, i.e., $c(e) \in [c^{min}, c^{min} + \alpha(c^{max} - c^{min})]$. The case $\alpha = 0$ corresponds to a pure greedy algorithm, while $\alpha = 1$ is equivalent to a random construction. The pseudo code in Figure 1.4 is a refinement of the greedy randomized contruction pseudo-code shown in Figure 1.2. It shows that the parameter α controls the amounts of greediness and randomness in the algorithm.

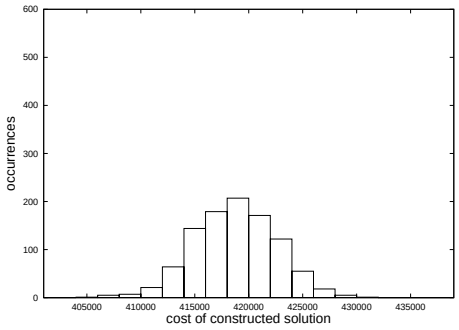
GRASP may be viewed as a repetitive sampling technique. Each iteration produces a sample solution from an unknown distribution, whose mean and variance are functions of the restrictive nature of the RCL. For example, if the RCL is restricted to a single element, then the same solution will be produced at all iterations. The variance of the distribution will be zero and the mean will be equal to the value of the greedy solution. If the RCL is allowed to have more elements, then many different solutions will be produced, implying a larger variance. Since greediness plays a smaller role in this case, the mean solution value should be worse. However, the value of the best solution found outperforms the mean value and very often is optimal. The histograms in Figure 1.5 illustrate this situation on an instance of MAXSAT with 100 variables and 850 clauses, depicting results obtained with 1000 independent constructions using the

```
procedure Greedy_Randomized_Construction(α, Seed)
1    Solution ← ∅;
2    Initialize the candidate set: C ← E;
3    Evaluate the incremental cost c(e) for all e ∈ C;
4    while C ≠ ∅ do
5        c^min ← min{c(e) | e ∈ C};
6        c^max ← max{c(e) | e ∈ C};
7        RCL ← {e ∈ C | c(e) ≤ c^min + α(c^max − c^min)};
8        Select an element s from the RCL at random;
9        Solution ← Solution ∪ {s};
10       Update the candidate set C;
11       Reevaluate the incremental costs c(e) for all e ∈ C;
12   end;
13   return Solution;
end Greedy_Randomized_Construction.
```

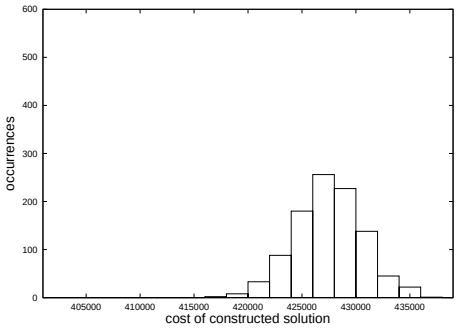
Figure 1.4 Refined pseudo-code of the construction phase.

first phase of the GRASP described in Resende et al. (1997) and Resende et al. (2000). Since this is a maximization problem, the purely greedy construction corresponds to $\alpha = 1$, whereas the random construction occurs with $\alpha = 0$. We notice that when the value of α increases from 0 to 1, the mean solution value increases towards the purely greedy solution value, while the variance approaches zero.

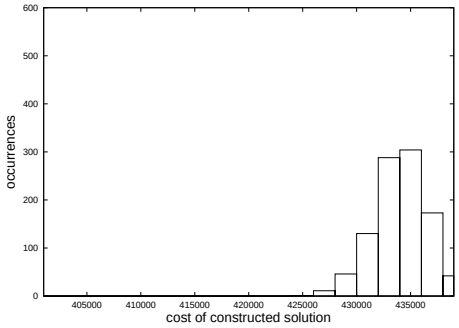
For each value of α, we present in Figure 1.6 histograms with the results obtained by applying local search to each of the 1000 constructed solutions. Figure 1.7 summarizes the overall results of this experiment in terms of solution diversity, solution quality, and computation time. We first observe that the larger the variance of the solution values obtained in the construction phase, the larger is the variance of the overall solution values, as shown in the top graph. The graph in the middle illustrates the relationship between the variance of the solution values and the average solution values, and how this affects the best solution found. It is unlikely that GRASP will find an optimal solution if the average solution value is low, even if there is a large variance in the overall solution values, such as is the case for $\alpha = 0$. On the other hand, if there is little variance in the overall solution values, it is also unlikely that GRASP will find an optimal solution, even if the average solution is high, as is the case for $\alpha = 1$. What often leads to good solutions are relatively high average solution values in the presence of a relatively large variance, such as is the case for $\alpha = 0.8$. The middle graph also shows that the distance between the average solution value and the value of the best solution found increases as the construction phase moves from more greedy to more random. This causes the average time taken by the local search to increase, as shown in the graph in the bottom. Very often, many GRASP solutions are generated in the same amount of time required for the local optimization procedure to converge from a single random start.



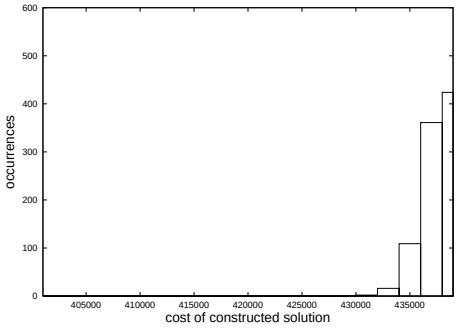
(a) RCL parameter alpha = 0.0 (random construction)



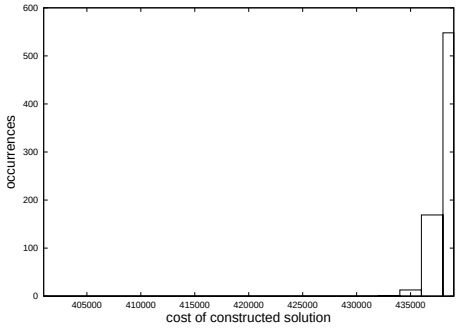
(b) RCL parameter alpha = 0.2



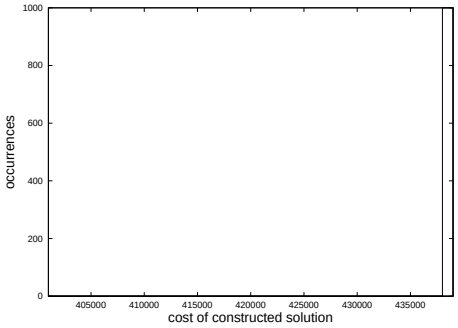
(c) RCL parameter alpha = 0.4



(d) RCL parameter alpha = 0.6

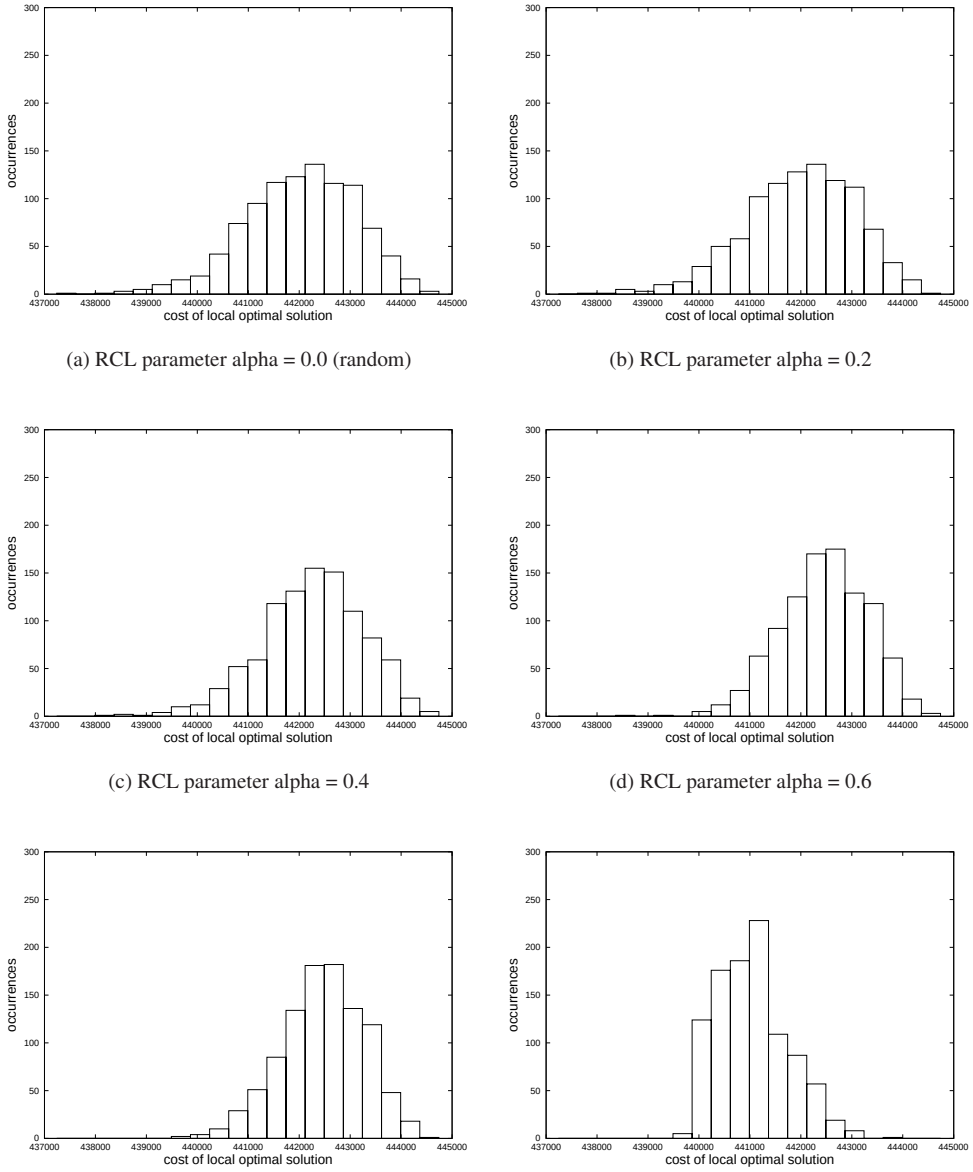


(e) RCL parameter alpha = 0.8



(f) RCL parameter alpha = 1.0 (greedy construction)

Figure 1.5 Distribution of construction phase solution values as a function of the RCL parameter α (1000 repetitions were recorded for each value of α).



(a) RCL parameter alpha = 0.0 (random)

(b) RCL parameter alpha = 0.2

(c) RCL parameter alpha = 0.4

(d) RCL parameter alpha = 0.6

(e) RCL parameter alpha = 0.8

(f) RCL parameter alpha = 1.0 (greedy)

Figure 1.6 Distribution of local search phase solution values as a function of the RCL parameter α (1000 repetitions for each value of α).

These results are illustrated in Table 1.1 and Figure 1.8, for another instance of MAXSAT where 1000 iterations were run. For each value of α ranging from 0 (purely random construction) to 1 (purely greedy construction), we give in Table 1.1 the average Hamming distance between each solution built at the end of the construction phase and the corresponding local optimum obtained after local search, the average number of moves from the first to the latter, the local search time in seconds, and the total processing time in seconds. Figure 1.8 summarizes the values observed for the total processing time and the local search time. We notice that both time measures considerably decrease as α tends to 1, approaching the purely greedy choice. In particular, we observe that the average local search time taken by $\alpha = 0$ (purely random) is approximately 2.5 times that taken in the case $\alpha = 0.9$ (almost greedy). In this example, two to three greedily constructed solutions can be investigated in the same time needed to apply local search to one single randomly constructed solution. The appropriate choice of the value of the RCL parameter α is clearly critical and relevant to achieve a good balance between computation time and solution quality.

Table 1.1 Average number of moves and local search time as a function of the RCL parameter α.

α	avg. distance	avg. moves	local search time (s)	total time (s)
0.0	12.487	12.373	18.083	23.378
0.1	10.787	10.709	15.842	20.801
0.2	10.242	10.166	15.127	19.830
0.3	9.777	9.721	14.511	18.806
0.4	9.003	8.957	13.489	17.139
0.5	8.241	8.189	12.494	15.375
0.6	7.389	7.341	11.338	13.482
0.7	6.452	6.436	10.098	11.720
0.8	5.667	5.643	9.094	10.441
0.9	4.697	4.691	7.753	8.941
1.0	2.733	2.733	5.118	6.235

Prais and Ribeiro (2000a) have shown that using a single fixed value for the value of RCL parameter α very often hinders finding a high-quality solution, which eventually could be found if another value was used. They proposed an extension of the basic GRASP procedure, which they call *Reactive GRASP*, in which the parameter α is self-tuned and its value is periodically modifed according with the quality of the solutions obtained recently. In particular, computational experiments on the problem of traffic assignment in communication satellites (Prais and Ribeiro, 2000b) have shown that Reactive GRASP found better solutions than the basic algorithm for many test instances. These results motivated the study of the behavior of GRASP for different strategies for the variation of the value of the RCL parameter α:

(R) α self tuned according with the Reactive GRASP procedure;

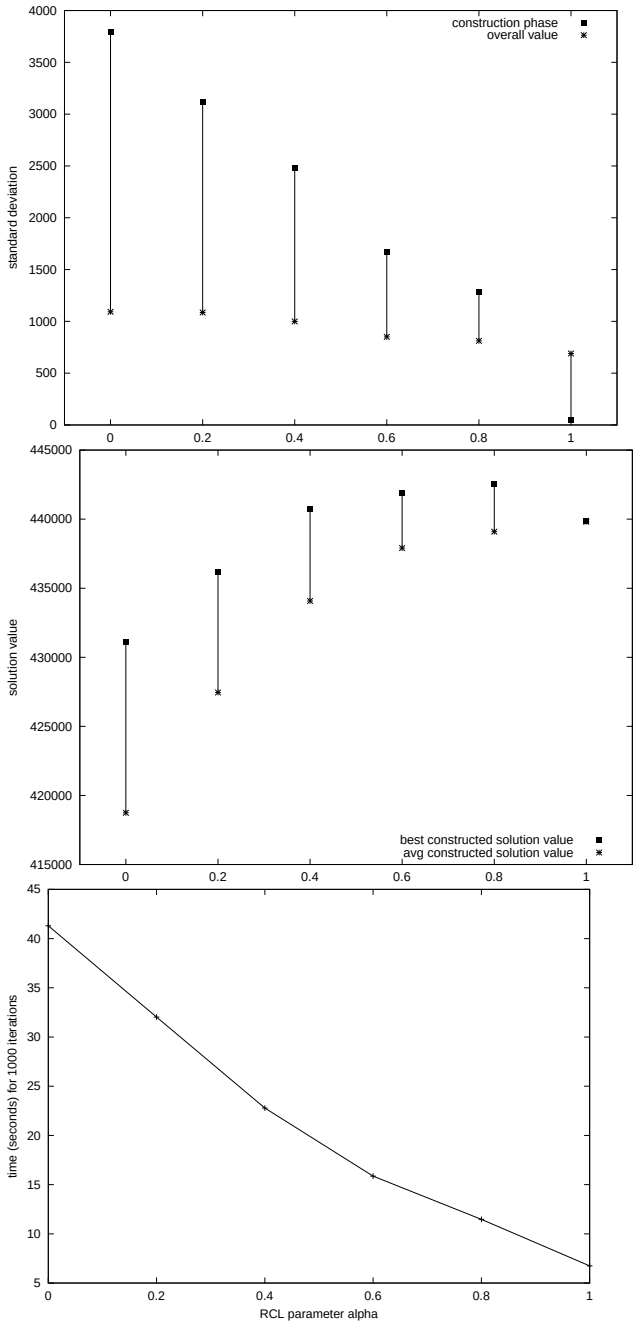


Figure 1.7 Standard deviation of the solution values found, best and average solution values found, and total processing time as a function of the RCL parameter α (1000 repetitions were recorded for each value of α).

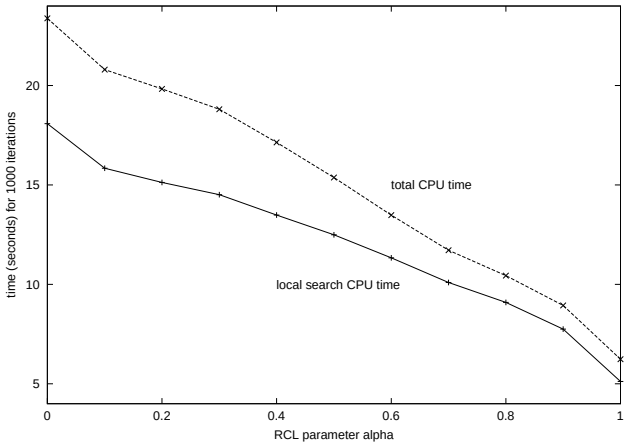


Figure 1.8 Total CPU time and local search CPU time as a function of the RCL parameter α (1000 repetitions for each value of α).

(E) α randomly chosen from a uniform discrete probability distribution;

(H) α randomly chosen from a decreasing non-uniform discrete probability distribution; and

(F) fixed value of α, close to the purely greedy choice.

We summarize the results obtained by the experiments reported in Prais and Ribeiro (1999) and Prais and Ribeiro (2000a). These four strategies were incorporated into the GRASP procedures developed for four different optimization problems: (P-1) matrix decomposition for traffic assignment in communication satellite (Prais and Ribeiro, 2000b), (P-2) set covering (Feo and Resende, 1989), (P-3) weighted MAX-SAT (Resende et al., 1997; 2000), and (P-4) graph planarization (Resende and Ribeiro, 1997; Ribeiro and Resende, 1999). Let $\Psi = \{\alpha_1, \ldots, \alpha_m\}$ be the set of possible values for the parameter α for the first three strategies. The strategy for choosing and self-tuning the value of α in the case of the Reactive GRASP procedure (R) is described later in Section 1.3. In the case of the strategy based on using the discrete uniform distribution (E), all choice probabilities are equal to $1/m$. The third case corresponds to the a hybrid strategy (H), in which we typically consider $p(\alpha = 0.1) = 0.5$, $p(\alpha = 0.2) = 0.25$, $p(\alpha = 0.3) = 0.125$, $p(\alpha = 0.4) = 0.03$, $p(\alpha = 0.5) = 0.03$, $p(\alpha = 0.6) = 0.03$, $p(\alpha = 0.7) = 0.01$, $p(\alpha = 0.8) = 0.01$, $p(\alpha = 0.9) = 0.01$, and $p(\alpha = 1.0) = 0.005$. Finally, in the last strategy (F), the value of α is fixed as recommended in the original reference where this parameter was tuned for each problem. A subset of the literature instances was considered for each class of test problems. The results reported in Prais and Ribeiro (2000a) are summarized in Table 1.2. For each problem, we first list the number of instances considered. Next, for each strategy, we give the number of times it found the best solution (hits), as well as the average CPU time (in seconds) on an IBM 9672 model R34. The number of iterations was fixed at 10,000.

Table 1.2 Computational results for different strategies for the variation of parameter α.

		R		*E*		*H*		*F*	
Problem	Total	hits	time	hits	time	hits	time	hits	time
P-1	36	34	579.0	35	358.2	32	612.6	24	642.8
P-2	7	7	1346.8	6	1352.0	6	668.2	5	500.7
P-3	44	22	2463.7	23	2492.6	16	1740.9	11	1625.2
P-4	37	28	6363.1	21	7292.9	24	6326.5	19	5972.0
Total	124	91		85		78		59	

Strategy (F) presented the shortest average computation times for three out the four problem types. It was also the one with the least variability in the constructed solutions and, in consequence, found the best solution the fewest times. The reactive strategy (R) is the one which most often found the best solutions, however, at the cost of computation times that are longer than those of some of the other strategies. The high number of hits observed by strategy (E) also illustrates the effectiveness of strategies based on the variation of the RCL parameter.

1.3 ALTERNATIVE CONSTRUCTION MECHANISMS

One possible shortcoming of the standard GRASP framework is the independence of its iterations, i.e., the fact that it does not learn from the history of solutions found in previous iterations. This is so because the basic algorithm discards information about any solution encountered that does not improve the incumbent. Information gathered from good solutions can be used to implement memory-based procedures to influence the construction phase, by modifying the selection probabilities associated with each element of the RCL. In this section, we consider enhancements and alternative techniques for the construction phase of GRASP. They include Reactive GRASP, cost perturbations in place of randomized selection, bias functions, memory and learning, and local search on partially constructed solutions.

1.3.1 Reactive GRASP

A first possible strategy to incorporate a learning mechanism in the memoryless construction phase of the basic GRASP is the Reactive GRASP procedure introduced in Section 1.2. In this case, the value of the RCL parameter α is not fixed, but instead is selected at each iteration from a discrete set of possible values. This selection is guided by the solution values found along the previous iterations. One way to accomplish this is to use the rule proposed in Prais and Ribeiro (2000b). Let $\Psi = \{\alpha_1, \ldots, \alpha_m\}$ be the set of possible values for α. The probabilities associated with the choice of each value are all initially made equal to $p_i = 1/m$, $i = 1, \ldots, m$. Furthermore, let z^* be the incumbent solution and let A_i be the average value of all solutions found using $\alpha = \alpha_i$, $i = 1, \ldots, m$. The selection probabilities are periodically reevaluated by taking $p_i = q_i / \sum_{j=1}^{m} q_j$, with $q_i = z^*/A_i$ for $i = 1, \ldots, m$. The value of q_i will be larger for

values of $\alpha = \alpha_i$ leading to the best solutions on average. Larger values of q_i correspond to more suitable values for the parameter α. The probabilities associated with these more appropriate values will then increase when they are reevaluated.

The reactive approach leads to improvements over the basic GRASP in terms of robustness and solution quality, due to greater diversification and less reliance on parameter tuning. In addition to the applications in Prais and Ribeiro (1999), Prais and Ribeiro (2000a), and Prais and Ribeiro (2000b), this approach has been used in power system transmission network planning (Binato and Oliveira, 2002) and in a capacitated location problem (Delmaire et al., 1999).

1.3.2 Cost perturbations

The idea of introducing some noise into the original costs is similar to that in the so-called "noising method" of Charon and Hudry (1993; 2002). It adds more flexibility into algorithm design and may be even more effective than the greedy randomized construction of the basic GRASP procedure, in circumstances where the construction algorithms are not very sensitive to randomization. This is indeed the case for the shortest-path heuristic of Takahashi and Matsuyama (1980), used as one of the main building blocks of the construction phase of the hybrid GRASP procedure proposed by Ribeiro et al. (2002) for the Steiner problem in graphs. Another situation where cost perturbations can be effective appears when no greedy algorithm is available for straight randomization. This happens to be the case of the hybrid GRASP developed by Canuto et al. (2001) for the prize-collecting Steiner tree problem, which makes use of the primal-dual algorithm of Goemans and Williamson (1996) to build initial solutions using perturbed costs.

In the case of the GRASP for the prize-collecting Steiner tree problem described in Canuto et al. (2001), a new solution is built at each iteration using node prizes updated by a perturbation function, according to the structure of the current solution. Two different prize perturbation schemes are used:

- *Perturbation by eliminations*: To enforce search diversification, the primal-dual algorithm used in the construction phase is driven to build a new solution without some of the nodes appearing in the solution constructed in the previous iteration. This is done by changing to zero the prizes of some persistent nodes, which appeared in the last solution built and remained at the end of the local search. A parameter α controls the fraction of the persistent nodes whose prizes are temporarily set to zero.

- *Perturbation by prize changes*: Another strategy to enforce the primal-dual algorithm to build different, but still good solutions, consists in introducing some noise into the node prizes, similarly to what is proposed in Charon and Hudry (1993; 2002), so as to change the objective function. For each node i, a perturbation factor $\beta(i)$ is randomly generated in the interval $[1-a, 1+a]$, where a is an implementation parameter. The prize associated with node i is temporarily changed to $\overline{\pi}(i) = \pi(i) \cdot \beta(i)$, where $\pi(i)$ is its original prize.

The cost perturbation methods used in the GRASP for the minimum Steiner tree problem described in Ribeiro et al. (2002) incorporate learning mechanisms associated

with intensification and diversification strategies, originally proposed in the context of tabu search. Let w_e denote the weight of edge e. Three distinct weight randomization methods (D, I, U) are applied. At a given GRASP iteration i, the modified weight w_e^i of each edge e is randomly selected from a uniform distribution between w_e and $r_i(e) \cdot w_e$, where the coefficient $r_i(e)$ depends on the selected weight randomization method applied at iteration i. Let $t_{i-1}(e)$ be the number of locally optimal solutions in which edge e appeared, after $i-1$ iterations of the hybrid GRASP procedure have been performed. Clearly, $0 \leq t_{i-1}(e) \leq i-1$. Table 1.3 displays how the coefficients $r_i(e)$ are computed by each randomization method.

Table 1.3 Maximum randomization coefficients.

Method	$r_i(e)$
D	$1.25 + 0.75 \cdot t_{i-1}(e)/(i-1)$
I	$2 - 0.75 \cdot t_{i-1}(e)/(i-1)$
U	2

In method D, values of the coefficients $r_i(e)$ are larger for edges which appeared more frequently in previously found local optima. This scheme leads to a diversification strategy, since more frequently used edges are likely to be penalized with stronger augmentations. Contrarily, method I is an intensification strategy penalizing less frequent edges with larger coefficients $r_i(e)$. Finally, the third randomization method U uses a uniform penalization strategy, independent of frequency information. The original weights without any penalization are used in the first three iterations, combined with three different construction heuristics. The weight randomization methods are then cyclically applied, one at each of the remaining iterations, starting with method I, next D, then U, then I again, and so on. The alternation between diversifying (method D) and intensifying (method I) iterations characterizes a strategic oscillation approach (Glover, 2000). The experimental results reported in Ribeiro et al. (2002) show that the strategy combining these three perturbation methods is more robust than any of them used isolated, leading to the best overall results on a quite broad mix of test instances with different characteristics. The hybrid GRASP with path-relinking using this cost perturbation strategy is among the most effective heuristics currently available for the Steiner problem in graphs.

1.3.3 Bias functions

In the construction procedure of the basic GRASP, the next element to be introduced in the solution is chosen at random from the candidates in the RCL. The elements of the RCL are assigned equal probabilities of being chosen. However, any probability distribution can be used to bias the selection toward some particular candidates. Another construction mechanism was proposed by Bresina (1996), where a family of such probability distributions is introduced. They are based on the rank $r(\sigma)$ assigned

to each candidate element σ, according to its value of the greedy function. Several bias functions are introduced, such as:

- random bias: $\texttt{bias}(r) = 1$;
- linear bias: $\texttt{bias}(r) = 1/r$;
- log bias: $\texttt{bias}(r) = \log^{-1}(r+1)$;
- exponential bias: $\texttt{bias}(r) = e^{-r}$; and
- polynomial bias of order n: $\texttt{bias}(r) = r^{-n}$.

Let $r(\sigma)$ denote the rank of element σ and let $\texttt{bias}(r(\sigma))$ be one of the bias function defined above. Once these values have been evaluated for all elements of the RCL, the probability $\pi(\sigma)$ of selecting element σ is

$$\pi(\sigma) = \frac{\texttt{bias}(r(\sigma))}{\sum_{\sigma' \in \mathrm{RCL}} \texttt{bias}(r(\sigma'))}. \tag{1.1}$$

The evaluation of these bias functions may be restricted to the elements of the RCL. Bresina's selection procedure restricted to elements of the RCL was used in Binato et al. (2002). Note that the standard GRASP uses a random bias function.

1.3.4 Intelligent construction: memory and learning

Fleurent and Glover (1999) observed that the basic GRASP does not use long-term memory (information gathered in previous iterations) and proposed a long-term memory scheme to address this issue in multi-start heuristics. Long-term memory is one of the fundamentals on which tabu search relies.

Their scheme maintains a pool of elite solutions to be used in the construction phase. To become an elite solution, a solution must be either better than the best member of the pool, or better than its worst member and sufficiently different from the other solutions in the pool. For example, one can count identical solution vector components and set a threshold for rejection.

A *strongly determined variable* is one that cannot be changed without eroding the objective or changing significantly other variables. A *consistent variable* is one that receives a particular value in a large portion of the elite solution set. Let $I(e)$ be a measure of the strongly determined and consistent features of solution element $e \in E$. Then, $I(e)$ becomes larger as e appears more often in the pool of elite solutions. The intensity function $I(e)$ is used in the construction phase as follows. Recall that $c(e)$ is the greedy function, i.e. the incremental cost associated with the incorporation of element $e \in E$ into the solution under construction. Let $K(e) = F(c(e), I(e))$ be a function of the greedy and the intensification functions. For example, $K(e) = \lambda c(e) + I(e)$. The intensification scheme biases selection from the RCL to those elements $e \in E$ with a high value of $K(e)$ by setting its selection probability to be $p(e) = K(e)/\sum_{s \in \mathrm{RCL}} K(s)$.

The function $K(e)$ can vary with time by changing the value of λ, e.g. initially λ may be set to a large value that is decreased when diversification is called for. Procedures for changing the value of λ are given by Fleurent and Glover (1999) and Binato et al. (2002).

1.3.5 POP in construction

The Proximate Optimality Principle (POP) is based on the idea that "good solutions at one level are likely to be found 'close to' good solutions at an adjacent level" (Glover and Laguna, 1997). Fleurent and Glover (1999) provided a GRASP interpretation of this principle. They suggested that imperfections introduced during steps of GRASP construction can be "ironed-out" by applying local search during (and not only at the end of) the GRASP construction phase.

Because of efficiency considerations, a practical implementation of POP to GRASP is to apply local search during a few points in the construction phase and not during each construction iteration. In Binato et al. (2002), local search is applied after 40% and 80% of the construction moves have been taken, as well as at the end of the construction phase.

1.4 PATH-RELINKING

Path-relinking is another enhancement to the basic GRASP procedure, leading to significant improvements in solution quality. Path-relinking was originally proposed by Glover (1996) as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search (Glover, 2000; Glover and Laguna, 1997; Glover et al., 2000). Starting from one or more elite solutions, paths in the solution space leading towards other elite solutions are generated and explored in the search for better solutions. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions. Path-relinking may be viewed as a strategy that seeks to incorporate attributes of high quality solutions, by favoring these attributes in the selected moves.

The use of path-relinking within a GRASP procedure, as an intensification strategy applied to each locally optimal solution, was first proposed by Laguna and Martí (1999). It was followed by several extensions, improvements, and successful applications (Aiex et al., 2000; Canuto et al., 2001; Resende and Ribeiro, 2001; Ribeiro et al., 2002). Two basic strategies are used:

- path-relinking is applied as a post-optimization step to all pairs of elite solutions; and
- path-relinking is applied as an intensification strategy to each local optimum obtained after the local search phase.

Applying path-relinking as an intensification strategy to each local optimum seems to be more effective than simply using it as a post-optimization step. In this context, path-relinking is applied to pairs (x_1, x_2) of solutions, where x_1 is the locally optimal solution obtained after local search and x_2 is one of a few elite solutions randomly chosen from a pool with a limited number `Max_Elite` of elite solutions found along the search. The pool is originally empty. Each locally optimal solution obtained by local search is considered as a candidate to be inserted into the pool if it is sufficiently different from every other solution currently in the pool. If the pool already has `Max_Elite` solutions and the candidate is better than the worst of them, then the former replaces the latter. If the pool is not full, the candidate is simply inserted.

The algorithm starts by computing the symmetric difference $\Delta(x_1, x_2)$ between x_1 and x_2, resulting in the set of moves which should be applied to one of them (the initial solution) to reach the other (the guiding solution). Starting from the initial solution, the best move from $\Delta(x_1, x_2)$ still not performed is applied to the current solution, until the guiding solution is attained. The best solution found along this trajectory is also considered as a candidate for insertion in the pool and the incumbent is updated. Several alternatives have been considered and combined in recent implementations:

- do not apply path-relinking at every GRASP iteration, but only periodically;
- explore two different trajectories, using first x_1, then x_2 as the initial solution;
- explore only one trajectory, starting from either x_1 or x_2; and
- do not follow the full trajectory, but instead only part of it (truncated path-relinking).

All these alternatives involve the trade-offs between computation time and solution quality. Ribeiro et al. (2002) observed that exploring two different trajectories for each pair (x_1, x_2) takes approximately twice the time needed to explore only one of them, with very marginal improvements in solution quality. They have also observed that if only one trajectory is to be investigated, better solutions are found when path-relinking starts from the best among x_1 and x_2. Since the neighborhood of the initial solution is much more carefully explored than that of the guiding one, starting from the best of them gives the algorithm a better chance to investigate in more detail the neighborhood of the most promising solution. For the same reason, the best solutions are usually found closer to the initial solution than to the guiding solution, allowing pruning the relinking trajectory before the latter is reached.

Detailed computational results illustrating the trade-offs between these strategies for the problem of routing private virtual circuits in frame-relay services are reported by Resende and Ribeiro (2001). In this case, the set of moves corresponding to the symmetric difference $\Delta(x_1, x_2)$ between any pair (x_1, x_2) of solutions is the subset of private virtual circuits routed through different routes (i.e., using different edges) in x_1 and x_2. We summarize below some of these results, obtained on an SGI Challenge computer (with 28 196-MHz MIPS R10000 processors) with 7.6 Gb of memory. We considered four variants of the GRASP and path-relinking schemes previously discussed:

- `G`: This variant is a pure GRASP with no path-relinking.
- `GPRf`: This variant adds to `G` a one-way (forward) path-relinking starting from a locally optimal solution and using a randomly selected elite solution as the guiding solution.
- `GPRb`: This variant adds to `G` a one way (backwards) path-relinking starting from a randomly selected elite solution and using a locally optimal solution as the guiding solution.
- `GPRfb`: This variant combines `GPRf` and `GPRb`, performing path-relinking in both directions.

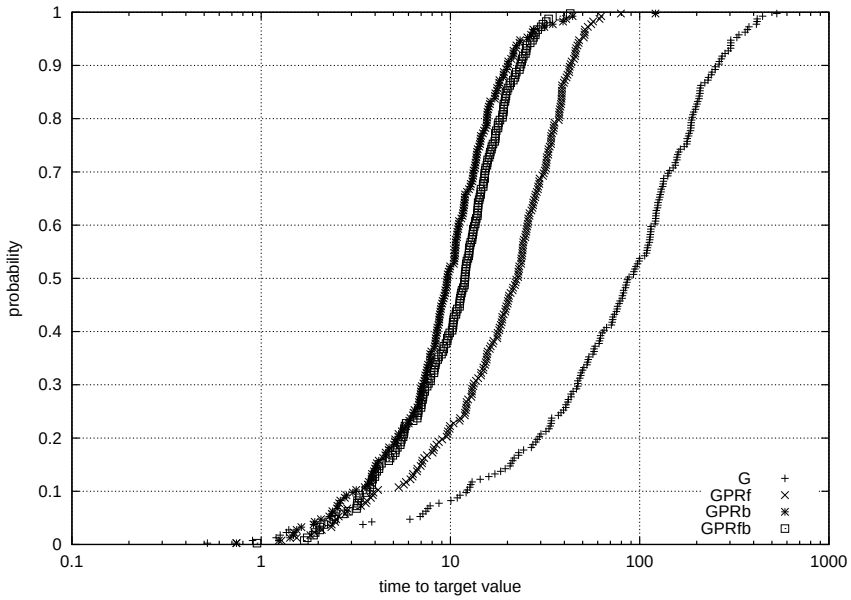


Figure 1.9 Empirical distributions of time to target solution value for GRASP, GRASP with forward path-relinking, GRASP with backwards path-relinking, and GRASP with back and forward path-relinking for instance `att`.

These variants are evaluated and compared in terms of their tradeoffs between computation time and solution quality.

To study the effect of path-relinking on GRASP, we compared the four variants on two instances: `att` and `fr750a`, see Resende and Ribeiro (2001) for details. Two hundred independent runs for each variant were performed for each problem. Execution was terminated when a solution of value less than or equal to a given parameter value `look4` was found. The sub-optimal values chosen for this parameter were such that the slowest variant could terminate in a reasonable amount of computation time. Empirical probability distributions for the time to target solution value are plotted in Figures 1.9 and 1.10. To plot the empirical distribution for each algorithm and each instance, we associate with the i-th smallest running time t_i a probability $p_i = (i - \frac{1}{2})/200$, and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 200$. Due to the time taken by the pure GRASP procedure, we limited its plot in Figure 1.10 to 60 points.

These plots show a similar relative behavior of the four variants on the two instances. Since instance `fr750a` is harder for all variants and the associated computation times are longer, its plot is more discerning. For a given computation time, the probability of finding a solution at least as good as the target value increases from `G` to `GPRf`, from `GPRf` to `GPRfb`, and from `GPRfb` to `GPRb`. For example, there is a 9.25% probability for `GPRfb` to find a target solution value in less than 100 seconds, while this probability increases to 28.75% for `GPRb`. For `G`, there is a 8.33% probability of finding a target solution value within 2000 seconds, while for `GPRf` this probability in-

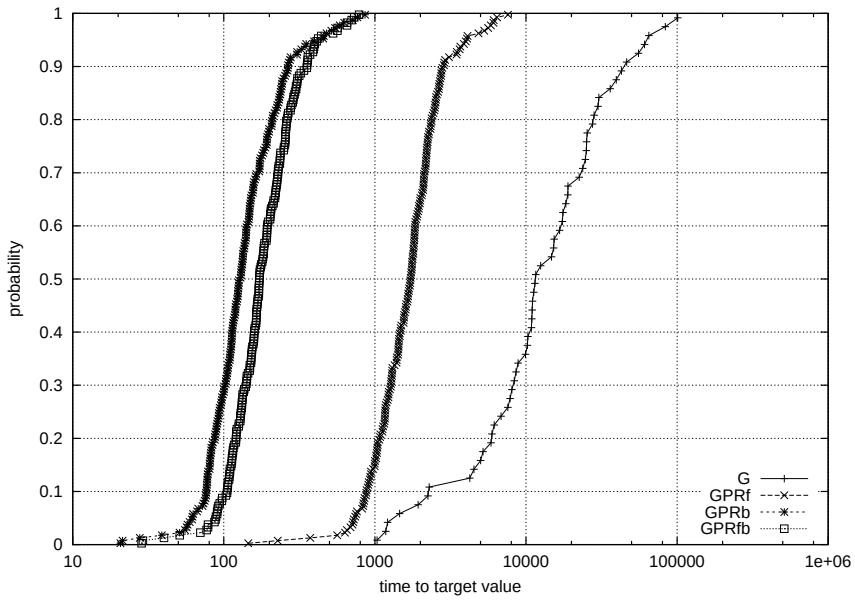


Figure 1.10 Empirical distributions of time to target solution value for GRASP, GRASP with forward path-relinking, GRASP with backwards path-relinking, and GRASP with back and forward path-relinking for instance `fr750a`.

creases to 65.25%. `GPRb` finds a target solution value in at most 129 seconds with 50% probability. For the same probability, this time increases to 172, 1727, and 10933 seconds, respectively, for variants `GPRfb`, `GPRf`, and `G`. In accordance with these results, variant `GPRb`, which does path-relinking backwards from an elite solution to a locally optimal solution, seems to be the most effective, confirming the preliminary findings reported in Ribeiro et al. (2002). To further illustrate the behavior of GRASP and path-relinking, we depict in Figure 1.11 four plots representing the behavior of variant `GPRb` (GRASP with backwards path-relinking) on instance `att` with the variation of the target solution value. As before, 200 runs were performed for each target value decreasing from 126,600 to 126,000 by steps of 200. A similar behavior was observed for all other variants, with or without path-relinking, as well as for other instances and classes of test problems.

As a final experiment, once again we made use of the different GRASP variants for the problem of routing private virtual circuits to illustrate the effect of path-relinking in improving the solutions obtained by a pure GRASP approach, with only the construction and local search phases. This experiment was also performed using the same SGI Challenge computer (with 28 196-MHz MIPS R10000 processors) with 7.6 Gb of memory. For each of ten different seeds, we ran twice each variant for instance `att`, enforcing two different time limits: 10 seconds and 100 seconds of processing time. The numerical results are reported in Table 1.4. For each variant and for each time limit, we give the average and the best solution values over the ten runs. We first

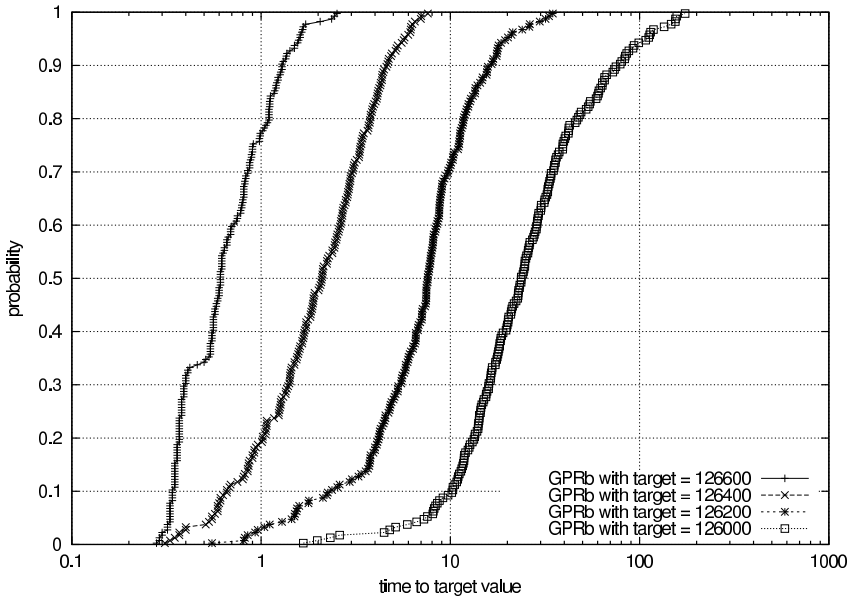


Figure 1.11 Empirical distributions of time to target solution value for GRASP with backwards path-relinking for instance `att` and different target values (`look4`).

note that both versions with backwards path-relinking performed systematically better, since they found better solutions for both time limits. Variants `GPRb` (GRASP with backwards path-relinking) and `GPRfb` (GRASP with path-relinking in both directions) showed similar behaviors, as it could be anticipated from the empirical probability distributions depicted in Figure 1.9. Variant `GPRb` obtained better results (in terms of both the average and the best solution values found) within the time limit of 10 seconds, while variant `GPRfb` performed better for the time limit of 100 seconds. In the first case, `GPRb` found the best solution among the two variants in seven runs, while `GPRfb` did better for only two runs. However, when the time limit was increased to 100 seconds, `GPRb` found the best solutions in four runs, while `GPRfb` did better for five runs.

Table 1.4 Solution values within fixed time limits over ten runs for instance `att`.

	10 seconds		100 seconds	
Variant	best	average	best	average
`GPR`	126602.883	126694.666	126227.678	126558.293
`GPRf`	126301.118	126578.323	126082.790	126228.798
`GPRb`	125960.336	126281.156	125665.785	125882.605
`GPRfb`	125961.118	126306.736	125646.460	125850.396

Path-relinking is a quite effective strategy to introduce memory in GRASP, leading to very robust implementations. The results reported above can be further illustrated by those obtained with the hybrid GRASP with path-relinking algorithm for the Steiner problem in graphs described in Ribeiro et al. (2002), which in particular improved the best known solutions for 33 out of the 41 still open problems in series i640 of the SteinLib repository (Voss et al., 2001) on May 1st, 2001.

1.5 EXTENSIONS

In this section, we comment on some extensions, implementation strategies, and hybrids of GRASP.

The use of hashing tables to avoid cycling in conjunction with tabu search was proposed by Woodruff and Zemel (1993). A similar approach was later explored by Ribeiro et al. (1997) in their tabu search algorithm for query optimization in relational databases. In the context of GRASP implementations, hashing tables were first used by Martins et al. (2000) in their multineighborhood heuristic for the Steiner problem in graphs, to avoid the application of local search to solutions already visited in previous iterations.

Filtering strategies have also been used to speed up the iterations of GRASP, see e.g. Feo et al. (1994), Martins et al. (2000), and Prais and Ribeiro (2000b). In these cases, local search is not applied to all solutions obtained at the end of the construction phase, but instead only to some promising unvisited solutions, defined by a threshold with respect to the incumbent.

Almost all randomization effort in the basic GRASP algorithm involves the construction phase. Local search stops at the first local optimum. On the other hand, strategies such as VNS (Variable Neighborhood Search), proposed by Hansen and Mladenović Hansen and Mladenović (2002); Mladenović and Hansen (1997), rely almost entirely on the randomization of the local search to escape from local optima. With respect to this issue, GRASP and variable neighborhood strategies may be considered as complementary and potentially capable of leading to effective hybrid methods. A first attempt in this direction was done by Martins et al. Martins et al. (2000). The construction phase of their hybrid heuristic for the Steiner problem in graphs follows the greedy randomized strategy of GRASP, while the local search phase makes use of two different neighborhood structures as a VND procedure Hansen and Mladenović (2002); Mladenović and Hansen (1997). Their heuristic was later improved by Ribeiro et al. Ribeiro et al. (2002), one of the key components of the new algorithm being another strategy for the exploration of different neighborhoods. Ribeiro and Souza Ribeiro and Souza (2002) also combined GRASP with VND in a hybrid heuristic for the degree-constrained minimum spanning tree problem. Festa et al. Festa et al. (2001) studied different variants and combinations of GRASP and VNS for the MAX-CUT problem, finding and improving the best known solutions for some open instances from the literature.

GRASP has also been used in conjunction with genetic algorithms. Basically, the greedy randomized strategy used in the construction phase of a GRASP is applied to generate the initial population for a genetic algorithm. We may cite e.g. the genetic algorithm of Ahuja et al. Ahuja et al. (2000) for the quadratic assignment problem,

which makes use of the GRASP proposed by Li et al. Li et al. (1994) to create the initial population of solutions. A similar approach was used by Armony et al. Armony et al. (2000), with the initial population made up by both randomly generated solutions and those built by a GRASP.

The hybridization of GRASP with tabu search was first studied by Laguna and González-Velarde Laguna and González-Velarde (1991). Delmaire et al. Delmaire et al. (1999) considered two approaches. In the first, GRASP is applied as a powerful diversification strategy in the context of a tabu search procedure. The second approach is an implementation of the Reactive GRASP algorithm presented in Section 1.3.1, in which the local search phase is strengthened by tabu search. Results reported for the capacitated location problem show that the hybrid approaches perform better than the isolated methods previously used. Two two-stage heuristics are proposed in Abdinnour-Helm and Hadley (2000) for solving the multi-floor facility layout problem. GRASP/TS applies a GRASP to find the initial layout and tabu search to refine it.

1.6 PARALLEL GRASP

Even though parallelism is not yet systematically used to speed up or to improve the effectiveness of metaheuristics, parallel implementations are very robust and abound in the literature; see e.g. Cung et al. Cung et al. (2002) for a recent survey.

Most parallel implementations of GRASP follow the *multiple-walk independent thread* strategy, based on the distribution of the iterations over the processors Alvim (1998); Alvim and Ribeiro (1998); Feo et al. (1994); Li et al. (1994); Martins et al. (1999; 1998); Murphey et al. (1998); Pardalos et al. (1995; 1996). In general, each search thread has to perform $\texttt{Max\_Iterations}/p$ iterations, where p and $\texttt{Max\_Iterations}$ are, respectively, the number of processors and the total number of iterations. Each processor has a copy of the sequential algorithm, a copy of the problem data, and an independent seed to generate its own pseudorandom number sequence. To avoid that the processors find the same solutions, each of them must use a different sequence of pseudorandom numbers. A single global variable is required to store the best solution found over all processors. One of the processors acts as the master, reading and distributing problem data, generating the seeds which will be used by the pseudorandom number generators at each processor, distributing the iterations, and collecting the best solution found by each processor. Since the iterations are completely independent and very little information is exchanged, linear speedups are easily obtained provided that no major load imbalance problems occur. The iterations may be evenly distributed over the processors or according with their demands, to improve load balancing.

Martins et al. Martins et al. (1998) implemented a parallel GRASP for the Steiner problem in graphs. Parallelization is achieved by the distribution of 512 iterations over the processors, with the value of the RCL parameter α randomly chosen in the interval $[0.0, 0.3]$ at each iteration. The algorithm was implemented in C on an IBM SP-2 machine with 32 processors, using the MPI library for communication. The 60 problems from series C, D, and E of the OR-Library Beasley (1990) have been used for the computational experiments. The parallel implementation obtained 45 optimal solutions over the 60 test instances. The relative deviation with respect to the optimal

value was never larger than 4%. Almost-linear speedups observed for 2, 4, 8, and 16 processors with respect to the sequential implementation are illustrated in Figure 1.12.

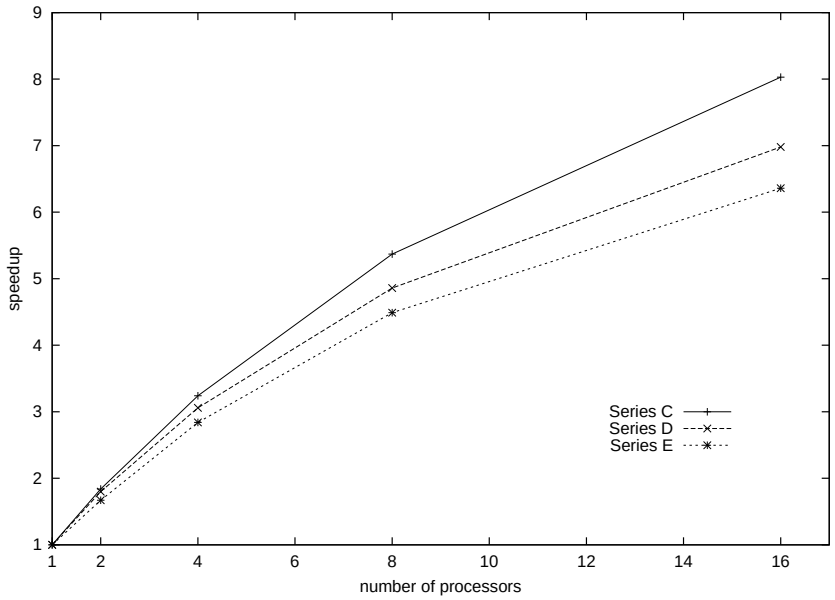


Figure 1.12 Average speedups on 2, 4, 8, and 16 processors.

Path-relinking may also be used in conjunction with parallel implementations of GRASP. In the case of the multiple-walk independent-thread implementation described by Aiex et al. Aiex et al. (2000) for the 3-index assignment problem, each processor applies path-relinking to pairs of elite solutions stored in a local pool. Computational results using MPI on an SGI Challenge computer with 28 R10000 processors showed linear speedups.

Alvim and Ribeiro Alvim (1998); Alvim and Ribeiro (1998) have shown that multiple-walk independent-thread approaches for the parallelization of GRASP may benefit much from load balancing techniques, whenever heterogeneous processors are used or if the parallel machine is simultaneously shared by several users. In this case, almost-linear speedups may be obtained with a heterogeneous distribution of the iterations over the p processors in $q \geq p$ packets. Each processor starts performing one packet of $\lceil \texttt{Max\_Iterations}/q \rceil$ iterations and informs the master when it finishes its packet of iterations. The master stops the execution of each slave processor when there are no more iterations to be performed and collects the best solution found. Faster or less loaded processors will perform more iterations than the others. In the case of the parallel GRASP implemented for the problem of traffic assignment described in Prais and Ribeiro (2000b), this dynamic load balancing strategy allowed reductions in the elapsed times of up to 15% with respect to the times observed for the static strategy, in which the iterations were uniformly distributed over the processors.

The efficiency of multiple-walk independent-thread parallel implementations of metaheuristics, based on running multiple copies of the same sequential algorithm, has been addressed by some authors. A given target value τ for the objective function is broadcasted to all processors which independently execute the sequential algorithm. All processors halt immediately after one of them finds a solution with value at least as good as τ. The speedup is given by the ratio between the times needed to find a solution with value at least as good as τ, using respectively the sequential algorithm and the parallel implementation with p processors. Some care is needed to ensure that no two iterations start with identical random number generator seeds. These speedups are linear for a number of metaheuristics, including simulated annealing Dodd (1990); Osborne and Gillett (1991); iterated local search algorithms for the traveling salesman problem Eikelder et al. (1996); tabu search, provided that the search starts from a local optimum Battiti and Tecchiolli (1992); Taillard (1991); and WalkSAT Selman et al. (1994) on hard random 3-SAT problems Hoos and Stützle (1999). This observation can be explained if the random variable *time to find a solution within some target value* is exponentially distributed, as indicated by the following proposition Verhoeven and Aarts (1995):

Proposition 1: Let $P_\rho(t)$ be the probability of not having found a given target solution value in t time units with ρ independent processes. If $P_1(t) = e^{-t/\lambda}$ with $\lambda \in \mathbf{R}^+$, corresponding to an exponential distribution, then $P_\rho(t) = e^{-\rho t/\lambda}$.

This proposition follows from the definition of the exponential distribution. It implies that the probability $1 - e^{-\rho t/\lambda}$ of finding a solution within a given target value in time ρt with a sequential algorithm is equal to the probability of finding a solution at least as good as that in time t using ρ independent parallel processors. Hence, it is possible to achieve linear speedups in the time to find a solution within a target value by multiple independent processors. An analogous proposition can be stated for a two parameter (shifted) exponential distribution:

Proposition 2: Let $P_\rho(t)$ be the probability of not having found a given target solution value in t time units with ρ independent processors. If $P_1(t) = e^{-(t-\mu)/\lambda}$ with $\lambda \in \mathbf{R}^+$ and $\mu \in \mathbf{R}^+$, corresponding to a two parameter exponential distribution, then $P_\rho(t) = e^{-\rho(t-\mu)/\lambda}$.

Analogously, this proposition follows from the definition of the two-parameter exponential distribution. It implies that the probability of finding a solution within a given target value in time ρt with a sequential algorithm is equal to $1 - e^{-(\rho t-\mu)/\lambda}$, while the probability of finding a solution at least as good as that in time t using ρ independent parallel processors is $1 - e^{-\rho(t-\mu)/\lambda}$. If $\mu = 0$, then both probabilities are equal and correspond to the non-shifted exponential distribution. Furthermore, if $\rho\mu \ll \lambda$, then the two probabilities are approximately equal and it is possible to approximately achieve linear speedups in the time to find a solution within a target value using multiple independent processors.

Aiex et al. Aiex et al. (2002) have shown experimentally that the solution times for GRASP also have this property, showing that they fit a two-parameter exponential distribution. Figure 1.13 illustrates this result, depicting the superimposed empirical and theoretical distributions observed for one of the cases studied along the computational experiments reported by the authors, which involved 2400 runs of GRASP

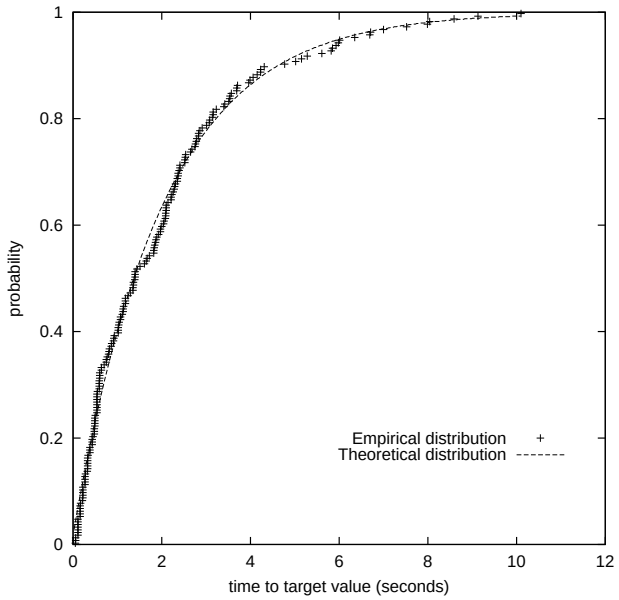


Figure 1.13 Superimposed empirical and theoretical distributions (times to target values measured in seconds on an SGI Challenge computer with 28 processors).

procedures for each of five different problems: maximum independent set Feo et al. (1994); Resende et al. (1998), quadratic assignment Li et al. (1994); Resende et al. (1996), graph planarization Resende and Ribeiro (1997); Ribeiro and Resende (1999), maximum weighted satisfiability Resende et al. (2000), and maximum covering Resende (1998). The same result still holds when GRASP is implemented in conjunction with a post-optimization path-relinking procedure Aiex et al. (2000).

In the case of *multiple-walk cooperative-thread* strategies, the search threads running in parallel exchange and share information collected along the trajectories they investigate. One expects not only to speed up the convergence to the best solution but, also, to find better solutions than independent-thread strategies. The most difficult aspect to be set up is the determination of the nature of the information to be shared or exchanged to improve the search, without taking too much additional memory or time to be collected. Cooperative-thread strategies may be implemented using path-relinking, by combining elite solutions stored in a central pool with the local optima found by each processor at the end of each GRASP iteration. Canuto et al. Canuto et al. (2001) used path-relinking to implement a parallel GRASP for the prize-collecting Steiner tree problem. Their strategy is truly cooperative, since pairs of elite solutions from a centralized unique central pool are distributed to the processors which perform path-relinking in parallel. Computational results obtained with an MPI implementation running on a cluster of 32 400-MHz Pentium II processors showed linear speedups, further illustrating the effectiveness of path-relinking procedures used in conjunction with GRASP to improve the quality of the solutions found by the latter.

1.7 APPLICATIONS

The first application of GRASP described in the literature concerns the set covering problem Feo and Resende (1989). The reader is referred to Festa and Resende Festa and Resende (2002) for an annotated bibliography of GRASP and its applications. We conclude this chapter by summarizing below some references focusing the main applications of GRASP to problems in different areas:

- routing Argüello et al. (1997); Atkinson (1998); Bard et al. (1998); Carreto and Baker (2002); Kontoravdis and Bard (1995);
- logic Deshpande and Triantaphyllou (1998); Pardalos et al. (1996); Resende and Feo (1996); Resende et al. (1997);
- covering and partition Areibi and Vannelli (1997); Argüello et al. (1996); Feo and Resende (1989); Ghosh (1996); Hammer and Rader, Jr. (2001);
- location Abdinnour-Helm and Hadley (2000); Delmaire et al. (1999); Klincewicz (1992); Urban (1998); Urban et al. (2000);
- minimum Steiner tree Canuto et al. (1999); Martins et al. (1999; 2000; 1998); Ribeiro et al. (2002);
- optimization in graphs Abello et al. (1999); Feo et al. (1994); Laguna et al. (1994); Pardalos et al. (1999); Resende (1998); Resende and Ribeiro (1997); Ribeiro and Resende (1999);
- assignment Feo and González-Velarde (1995); Fleurent and Glover (1999); Li et al. (1994); Liu et al. (2000); Mavridou et al. (1998); Murphey et al. (1998); Pardalos et al. (1995); Pitsoulis et al. (2001); Prais and Ribeiro (2000b);
- timetabling, scheduling, and manufacturing Bard and Feo (1989; 1991); Bard et al. (1996); Binato et al. (2002); De et al. (1994); Drexl and Salewski (1997); Feo and Bard (1989a;b); Feo et al. (1995; 1996; 1991); Klincewicz and Rajan (1994); Ríos-Mercado and Bard (1998; 1999); Yen et al. (2000);
- transportation Argüello et al. (1997); Feo and Bard (1989a); Feo and González-Velarde (1995);
- power systems Binato and Oliveira (2002);
- telecommunications Abello et al. (1999); Armony et al. (2000); Klincewicz (1992); Liu et al. (2000); Prais and Ribeiro (2000b); Resende (1998); Resende and Ribeiro (2001);
- graph and map drawing Fernández and Martí (1999); Laguna and Martí (1999); Resende and Ribeiro (1997); Ribeiro and Resende (1999); and
- VLSI Areibi and Vannelli (1997), among other areas of application.

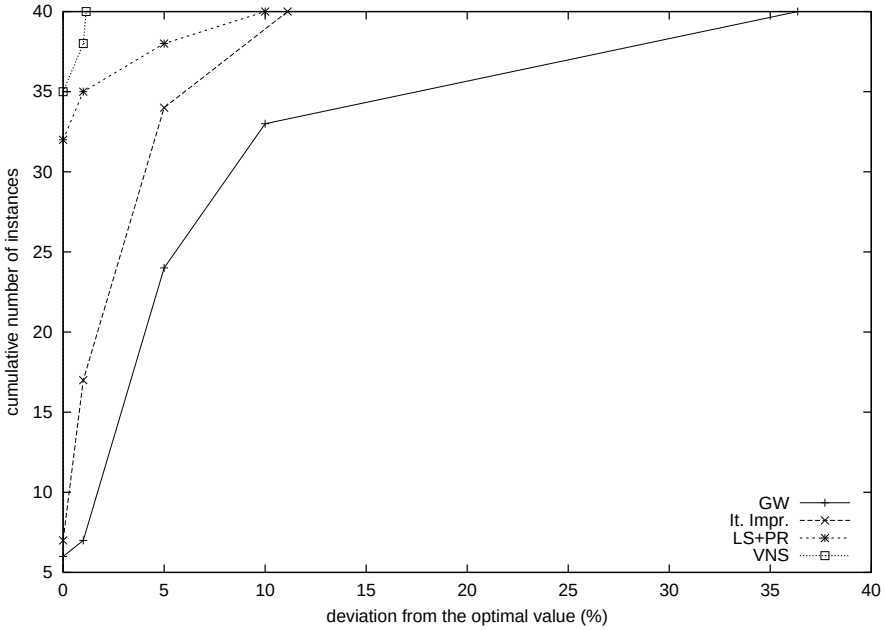


Figure 1.14 Performance of GW and successive variants of local search for Series C problems.

1.8 CONCLUDING REMARKS

The results described in this chapter reflect successful applications of GRASP to a large number of classical combinatorial optimization problems, as well as to those that arise in real-world situations in different areas of business, science, and technology.

We underscore the simplicity of implementation of GRASP, which makes use of simple building blocks: solution construction procedures and local search methods, which often are readily available. Contrary to what occurs with other metaheuristics, such as tabu search or genetic algorithms, which use a large number of parameters in their implementations, the basic version of GRASP requires the adjustment of a single parameter.

Recent developments, presented in this chapter, show that different extensions to the basic procedure allow further improvement to the solutions found by GRASP. Among these, we highlight: reactive GRASP, which automates the adjustments of the restricted candidate list parameter; variable neighborhoods, which permit accelerated and intensified local search; and path-relinking, which beyond allowing the implementation of intensification strategies based on the memory of elite solutions, opens the way for development of very effective cooperative parallel strategies.

These and other extensions make up a set of tools that can be added to simpler heuristics to find better-quality solutions. To illustrate the effect of additional extensions on solution quality, Figure 1.14 shows some results obtained for the prize-

collecting Steiner tree problem, as discussed in Canuto et al. (2001). We consider the 40 instances of series C. The lower curve represents the results obtained exclusively with the primal-dual constructive algorithm (GW) of Goemans and Williamson Goemans and Williamson (1996). The second curve shows the quality of the solutions produced with an additional local search (GW+LS), corresponding to the first iteration of GRASP. The third curve is associated with the results obtained after 500 iterations of GRASP with path-relinking (GRASP+PR). Finally, the top curve shows the results found by the complete algorithm, using variable neighborhood search as a post-optimization procedure (GRASP+PR+VNS). For a given relative deviation with respect to the optimal value, each curve indicates the number of instances for which the corresponding algorithm found a solution within that quality range. For example, we observe that the number of optimal solutions found goes from six, using only the constructive algorithm, to a total of 36, using the complete algorithm described in Canuto et al. (2001). The largest relative deviation with respect to the optimal value decreases from 36.4% in the first case, to only 1.1% for the complete algorithm. It is easy to see the contribution made by each additional extension.

Parallel implementations of GRASP are quite robust and lead to linear speedups both in independent and cooperative strategies. Cooperative strategies are based on the collaboration between processors using path-relinking and a global pool of elite solutions. This allows the use of more processors to find better solutions in less computation time.

Bibliography

S. Abdinnour-Helm and S.W. Hadley. Tabu search based heuristics for multi-floor facility layout. *International Journal of Production Research*, 38:365–383, 2000.

J. Abello, P.M. Pardalos, and M.G.C. Resende. On maximum clique problems in very large graphs. In J. Abello and J. Vitter, editors, *External Memory Algorithms and Visualization*, volume 50 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 199–130. American Mathematical Society, 1999.

R.K. Ahuja, J.B. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research*, 27:917–934, 2000.

R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with path-relinking for the three-index assignment problem. Technical report, AT&T Labs-Research, 2000.

R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8:343–373, 2002.

A.C. Alvim. Parallelization strategies for the metaheuristic GRASP. Master's thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Brazil, 1998. In Portuguese.

A.C. Alvim and C.C. Ribeiro. Load balancing for the parallelization of the GRASP metaheuristic. In *Proceedings of the X Brazilian Symposium on Computer Architecture*, pages 279–282, Búzios, 1998. In Portuguese.

S. Areibi and A. Vannelli. A GRASP clustering technique for circuit partitioning. In J. Gu and P.M. Pardalos, editors, *Satisfiability Problems*, volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 711–724. American Mathematical Society, 1997.

M.F. Argüello, J.F. Bard, and G. Yu. A GRASP for aircraft routing in response to groundings and delays. *Journal of Combinatorial Optimization*, 1:211–228, 1997.

M.F. Argüello, T.A. Feo, and O. Goldschmidt. Randomized methods for the number partitioning problem. *Computers and Operations Research*, 23:103–111, 1996.

M. Armony, J.C. Klincewicz, H. Luss, and M.B. Rosenwein. Design of stacked self-healing rings using a genetic algorithm. *Journal of Heuristics*, 6:85–105, 2000.

J.B. Atkinson. A greedy randomised search heuristic for time-constrained vehicle scheduling and the incorporation of a learning strategy. *Journal of the Operatinal Research Society*, 49:700–708, 1998.

J.F. Bard and T.A. Feo. Operations sequencing in discrete parts manufacturing. *Management Science*, 35:249–255, 1989.

J.F. Bard and T.A. Feo. An algorithm for the manufacturing equipment selection problem. *IIE Transactions*, 23:83–92, 1991.

J.F. Bard, T.A. Feo, and S. Holland. A GRASP for scheduling printed wiring board assembly. *IIE Transactions*, 28:155–165, 1996.

J.F. Bard, L. Huang, P. Jaillet, and M. Dror. A decomposition approach to the inventory routing problem with satellite facilities. *Transportation Science*, 32:189–203, 1998.

R. Battiti and G. Tecchiolli. Parallel biased search for combinatorial optimization: Genetic algorithms and tabu. *Microprocessors and Microsystems*, 16:351–367, 1992.

J.E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

S. Binato, W.J. Hery, D. Loewenstern, and M.G.C. Resende. A GRASP for job shop scheduling. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 59–79. Kluwer Academic Publishers, 2002.

S. Binato and G.C. Oliveira. A reactive GRASP for transmission network expansion planning. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 81–100. Kluwer Academic Publishers, 2002.

J.L. Bresina. Heuristic-biased stochastic sampling. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 271–278, Portland, 1996.

S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38:50–58, 2001.

S.A. Canuto, C.C. Ribeiro, and M.G.C. Resende. Local search with perturbations for the prize-collecting Steiner tree problem. In *Extended Abstracts of the Third Metaheuristics International Conference*, pages 115–119, Angra dos Reis, July 1999.

C. Carreto and B. Baker. A GRASP interactive approach to the vehicle routing problem with backhauls. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 185–199. Kluwer Academic Publishers, 2002.

I. Charon and O. Hudry. The noising method: A new method for combinatorial optimization. *Operations Research Letters*, 14:133–137, 1993.

I. Charon and O. Hudry. The noising methods: A survey. In C.C. Ribeiro and C.C. Ribeiro, editors, *Essays and Surveys in Metaheuristics*, pages 245–261. Kluwer Academic Publishers, 2002.

V.-D. Cung, S.L. Martins, C.C. Ribeiro, and C. Roucairol. Strategies for the parallel implementation of metaheuristics. In C.C. Ribeiro and C.C. Ribeiro, editors, *Essays and Surveys in Metaheuristics*, pages 263–308. Kluwer Academic Publishers, 2002.

P. De, J.B. Ghosj, and C.E. Wells. Solving a generalized model for con due date assignment and sequencing. *International Journal of Production Economics*, 34: 179–185, 1994.

H. Delmaire, J.A. Díaz, E. Fernández, and M. Ortega. Reactive GRASP and Tabu Search based heuristics for the single source capacitated plant location problem. *INFOR*, 37:194–225, 1999.

A.S. Deshpande and E. Triantaphyllou. A greedy randomized adaptive search procedure (GRASP) for inferring logical clauses from examples in polynomial time and some extensions. *Mathematical Computer Modelling*, 27:75–99, 1998.

N. Dodd. Slow annealing versus multiple fast annealing runs: An empirical investigation. *Parallel Computing*, 16:269–272, 1990.

A. Drexl and F. Salewski. Distribution requirements and compactness constraints in school timetabling. *European Journal of Operational Research*, 102:193–214, 1997.

H.T. Eikelder, M. Verhoeven, T. Vossen, and E. Aarts. A probabilistic analysis of local search. In I. Osman and J. Kelly, editors, *Metaheuristics: Theory and Applications*, pages 605–618. Kluwer Academic Publishers, 1996.

T.A. Feo and J.F. Bard. Flight scheduling and maintenance base planning. *Management Science*, 35:1415–1432, 1989a.

T.A. Feo and J.F. Bard. The cutting path and tool selection problem in computer-aided process planning. *Journal of Manufacturing Systems*, 8:17–26, 1989b.

T.A. Feo, J.F. Bard, and S. Holland. Facility-wide planning and scheduling of printed wiring board assembly. *Operations Research*, 43:219–230, 1995.

T.A. Feo and J.L. González-Velarde. The intermodal trailer assignment problem: Models, algorithms, and heuristics. *Transportation Science*, 29:330–341, 1995.

T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.

T.A. Feo, K. Sarathy, and J. McGahan. A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. *Computers and Operations Research*, 23:881–895, 1996.

T.A. Feo, K. Venkatraman, and J.F. Bard. A GRASP for a difficult single machine scheduling problem. *Computers and Operations Research*, 18:635–643, 1991.

E. Fernández and R. Martí. GRASP for seam drawing in mosaicking of aerial photographic maps. *Journal of Heuristics*, 5:181–197, 1999.

P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

P. Festa, M.G.C. Resende, P. Pardalos, and C.C. Ribeiro. GRASP and VNS for Max-Cut. In *Extended Abstracts of the Fourth Metaheuristics International Conference*, pages 371–376, Porto, July 2001.

C. Fleurent and F. Glover. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing*, 11: 198–204, 1999.

J.B. Ghosh. Computatinal aspects of the maximum diversity problem. *Operations Research Letters*, 19:175–181, 1996.

F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

F. Glover. Multi-start and strategic oscillation methods – Principles to exploit adaptive memory. In M. Laguna and J.L. Gonzáles-Velarde, editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pages 1–24. Kluwer, 2000.

F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.

F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

M.X. Goemans and D.P. Williamson. The primal dual method for approximation algorithms and its application to network design problems. In D. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 144–191. PWS Publishing Co., 1996.

P.L. Hammer and D.J. Rader, Jr. Maximally disjoint solutions of the set covering problem. *Journal of Heuristics*, 7:131–144, 2001.

P. Hansen and N. Mladenović. Developments of variable neighborhood search. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 415–439. Kluwer Academic Publishers, 2002.

J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.

H. Hoos and T. Stützle. Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. *Artificial Intelligence*, 112:213–232, 1999.

J.G. Klincewicz. Avoiding local optima in the *p*-hub location problem using tabu search and GRASP. *Annals of Operations Research*, 40:283–302, 1992.

J.G. Klincewicz and A. Rajan. Using GRASP to solve the component grouping problem. *Naval Research Logistics*, 41:893–912, 1994.

G. Kontoravdis and J.F. Bard. A GRASP for the vehicle routing problem with time windows. *ORSA Journal on Computing*, 7:10–23, 1995.

M. Laguna, T.A. Feo, and H.C. Elrod. A greedy randomized adaptive search procedure for the two-partition problem. *Operations Research*, 42:677–687, 1994.

M. Laguna and J.L. González-Velarde. A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent Manufacturing*, 2:253–260, 1991.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994.

X. Liu, P.M. Pardalos, S. Rajasekaran, and M.G.C. Resende. A GRASP for frequency assignment in mobile radio networks. In B.R. Badrinath, F. Hsu, P.M. Pardalos, and S. Rajasejaran, editors, *Mobile Networks and Computing*, volume 52 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 195–201. American Mathematical Society, 2000.

S.L. Martins, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Greedy randomized adaptive search procedures for the steiner problem in graphs. In P.M. Pardalos, S. Rajasejaran, and J. Rolim, editors, *Randomization Methods in Algorithmic Design*, volume 43 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 133–145. American Mathematical Society, 1999.

S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P. Pardalos. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization*, 17:267–283, 2000.

S.L. Martins, C.C. Ribeiro, and M.C. Souza. A parallel GRASP for the Steiner problem in graphs. In A. Ferreira and J. Rolim, editors, *Proceedings of IRREGULAR'98 – 5th International Symposium on Solving Irregularly Structured Problems in Parallel*, volume 1457 of *Lecture Notes in Computer Science*, pages 285–297. Springer-Verlag, 1998.

T. Mavridou, P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A GRASP for the biquadratic assignment problem. *European Journal of Operational Research*, 105: 613–621, 1998.

N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

R.A. Murphey, P.M. Pardalos, and L.S. Pitsoulis. A parallel GRASP for the data association multidimensional assignment problem. In P.M. Pardalos, editor, *Parallel Processing of Discrete Problems*, volume 106 of *The IMA Volumes in Mathematics and Its Applications*, pages 159–180. Springer-Verlag, 1998.

L. Osborne and B. Gillett. A comparison of two simulated annealing algorithms applied to the directed Steiner problem on networks. *ORSA Journal on Computing*, 3: 213–225, 1991.

P. M. Pardalos, T. Qian, and M. G. C. Resende. A greedy randomized adaptive search procedure for the feedback vertex set problem. *Journal of Combinatorial Optimization*, 2:399–412, 1999.

P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP implementation for the quadratic assignment problem. In A. Ferreira and J. Rolim, editors, *Parallel Algorithms for Irregularly Structured Problems – Irregular'94*, pages 115–133. Kluwer Academic Publishers, 1995.

P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP for MAX-SAT problems. *Lecture Notes in Computer Science*, 1184:575–585, 1996.

L.S. Pitsoulis, P.M. Pardalos, and D.W. Hearn. Approximate solutions to the turbine balancing problem. *European Journal of Operational Research*, 130:147–155, 2001.

M. Prais and C.C. Ribeiro. Parameter variation in GRASP implementations. In *Extended Abstracts of the Third Metaheuristics International Conference*, pages 375–380, Angra dos Reis, July 1999.

M. Prais and C.C. Ribeiro. Parameter variation in GRASP procedures. *Investigación Operativa*, 9:1–20, 2000a.

M. Prais and C.C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12: 164–176, 2000b.

M.G.C. Resende. Computing approximate solutions of the maximum covering problem using GRASP. *Journal of Heuristics*, 4:161–171, 1998.

M.G.C. Resende and T.A. Feo. A GRASP for satisfiability. In D.S. Johnson and M.A. Trick, editors, *Cliques, Coloring, and Satisfiability: The Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 499–520. American Mathematical Society, 1996.

M.G.C. Resende, T.A. Feo, and S.H. Smith. Algorithm 787: Fortran subroutines for approximate solution of maximum independent set problems using GRASP. *ACM Trans. Math. Software*, 24:386–394, 1998.

M.G.C. Resende, P.M. Pardalos, and Y. Li. Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software*, 22:104–118, 1996.

M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Approximate solution of weighted MAX-SAT problems using GRASP. In J. Gu and P.M. Pardalos, editors, *Satisfiability Problems*, volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 393–405. American Mathematical Society, 1997.

M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Fortran subroutines for computing approximate solutions of MAX-SAT problems using GRASP. *Discrete Applied Mathematics*, 100:95–113, 2000.

M.G.C. Resende and C.C. Ribeiro. A GRASP for graph planarization. *Networks*, 29: 173–189, 1997.

M.G.C. Resende and C.C. Ribeiro. A GRASP with path-relinking for private virtual circuit routing. Technical report, AT&T Labs Research, 2001.

C.C. Ribeiro and M.G.C. Resende. Algorithm 797: Fortran subroutines for approximate solution of graph planarization problems using GRASP. *ACM Transactions on Mathematical Software*, 25:342–352, 1999.

C.C. Ribeiro, C.D. Ribeiro, and R.S. Lanzelotte. Query optimization in distributed relational databases. *Journal of Heuristics*, 3:5–23, 1997.

C.C. Ribeiro and M.C. Souza. Variable neighborhood search for the degree constrained minimum spanning tree problem. *Discrete Applied Mathematics*, 118:43–54, 2002.

C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14:228–246, 2002.

R.Z. Ríos-Mercado and J.F. Bard. Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, pages 76–98, 1998.

R.Z. Ríos-Mercado and J.F. Bard. An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup costs. *Journal of Heuristics*, 5:57–74, 1999.

B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 337–343, Seattle, 1994. MIT Press.

E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 7:443–455, 1991.

H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Mathematica Japonica*, 24:573–577, 1980.

T.L. Urban. Solution procedures for the dynamic facility layout problem. *Annals of Operations Research*, pages 323–342, 1998.

T.L. Urban, W.-C. Chiang, and R.A. Russel. The integrated machine allocation and layout problem. *International Journal of Production Research*, pages 2913–2930, 2000.

M.G.A. Verhoeven and E.H.L. Aarts. Parallel local search. *Journal of Heuristics*, 1: 43–65, 1995.

S. Voss, A. Martin, and T. Koch. Steinlib testdata library, 2001. Online document at `http://elib.zib.de/steinlib/steinlib.html`, last visited on May 1st, 2001.

D.L. Woodruff and E. Zemel. Hashing vectors for tabu search. *Annals of Operations Research*, 41:123–137, 1993.

J. Yen, M. Carlsson, M. Chang, J.M. Garcia, and H. Nguyen. Constraint solving for inkjet print mask design. *Journal of Imaging Science and Technology*, 44:391–397, 2000.

2 GRASP WITH PATH-RELINKING

Mauricio G. C. Resende[1] and Celso C. Ribeiro[2]

[1]Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA
mgcr@research.att.com

[2]Department of Computer Science
Catholic University of Rio de Janeiro
Rio de Janeiro, RJ 22453-900 Brazil
celso@inf.puc-rio.br

Abstract: Path-relinking is a major enhancement to the basic greedy randomized adaptive search procedure (GRASP), leading to significant improvements in solution time and quality. Path-relinking adds a memory mechanism to GRASP by providing an intensification strategy that explores trajectories connecting GRASP solutions and the best elite solutions previously produced during the search. This paper reviews recent advances and applications of GRASP with path-relinking. A brief review of GRASP is given. This is followed by a description of path-relinking and how it is incorporated into GRASP. Several recent applications of GRASP with path-relinking are reviewed. The paper concludes with a discussion of extensions to this strategy, concerning in particular parallel implementations and applications of path-relinking with other metaheuristics.
Keywords: GRASP, metaheuristic, path-relinking.

2.1 INTRODUCTION

GRASP (*Greedy Randomized Adaptive Search Procedure*) is a metaheuristic for finding approximate solutions to combinatorial optimization problems formulated as

$$\min f(x) \text{ subject to } x \in X,$$

where $f(\cdot)$ is an objective function to be minimized and X is a discrete set of feasible solutions. It was first introduced by Feo and Resende (1989) in a paper describing a probabilistic heuristic for set covering. Since then, GRASP has experienced continued development (Feo and Resende, 1995; Resende and Ribeiro, 2003c) and has been applied in a wide range of problem areas (Festa and Resende, 2002).

2.1.1 Multi-start local search

GRASP can be thought of as a search method that repeatedly applies local search from different starting solutions in X. At each step of local search, the neighborhood $N(x)$ of the current solution x is searched for a solution $y \in N(x)$ such that $f(y) < f(x)$. If such an improving solution is found, it is made the current solution and another step of local search is done. If no improving solution is found, the procedure stops with x as a locally optimal solution.

An obvious initial solution for local search is a solution generated by a greedy algorithm. Greedy algorithms construct a solution one element at a time. For example, a tree is built one edge at a time; a schedule is built one operation at a time; a vertex partition is built one vertex at a time. At each step of a greedy algorithm, a set of candidate elements C contains all elements that can be added to extend the partial solution. Greedy algorithms make use of a greedy function $g(e)$ that measures the incremental cost of adding element $e \in C$ to the current partial solution. For a minimization problem, the element $e^* = \text{argmin}\{g(e) : e \in C\}$ is chosen to be added to the partial solution. The addition of e^* to the partial solution usually restricts the set of candidates elements, which is reflected by the reduction of its cardinality. The procedure ends when a complete solution is built, i.e. when $C = \emptyset$.

The drawback of using a greedy algorithm as an initial solution for local search is that if a deterministic rule is used to break ties, a greedy algorithm will produce a single solution and therefore local search can only be applied once. Even when a probabilistic tie breaking rule is used, the diversity of purely greedy solutions is usually low.

The other extreme is to repeatedly start local search from randomly generated solutions. Though this approach produces a high level of diversity in the starting solutions, the average quality of these random solutions is usually much worse than that of a greedy solution. Furthermore, the time local search takes to converge to a locally optimal solution is, on average, much longer than when a greedy initial solution is used.

GRASP blends greedy and random construction either by using greediness to build a restricted candidate list (RCL) and randomness to select an element from the list, or by using randomness to build the list and greediness for selection. Candidate elements $e \in C$ are sorted according to their greedy function value $g(e)$. In a cardinality-based RCL, the latter is made up by the k top-ranked elements. In a value-based construction, the RCL consists of the elements in the set $\{e \in C : g_* \leq g(e) \leq g_* + \alpha \cdot (g^* - g_*)\}$, where $g_* = \min\{g(e) : e \in C\}$, $g^* = \max\{g(e) : e \in C\}$, and α is a parameter satisfying $0 \leq \alpha \leq 1$. Since the best value for α is often difficult to determine, it is often assigned a random value for each GRASP iteration.

Algorithm 1 shows the pseudo-code for a pure greedy randomized adaptive search procedure. The value of the best solution is stored in f^* and $i_{\max}$ GRASP iterations are done. Each iteration consists of a greedy randomized construction phase, followed by a local search phase, starting from the greedy randomized solution. If the solution resulting from the local search improves the best solution so far, it is stored in x^*.

```
Data   : Number of iterations i_max
Result : Solution x* ∈ X
f* ← ∞;
for i = 1,...,i_max do
    x ← GreedyRandomizedConstruction();
    x ← LocalSearch(x);
    if f(x) < f* then
        f* ← f(x);
        x* ← x;
    end
end
```

Algorithm 1: A basic GRASP for minimization.

Figure 2.1 displays results for an instance of the maximum covering problem (Resende, 1998), showing the distribution of objective function values for the construction phase and the local search phase of a purely random multi-start algorithm (followed by local search) and a GRASP with the parameter α fixed at 0.85. In both plots, the iterates have been sorted by the objective function value of the solution found by local search. The plots show that the GRASP construction achieves a reasonably amount of diversity in terms of solution values, while producing starting solutions for local search that have much better objective function values. The objective function values are situated around 3.5 for the random construction and 9.7 for GRASP construction, while the value obtained by local search are around 9.9. Consequently, the local search times are much smaller for GRASP than for the purely random multi-start algorithm.

Figure 2.2 shows, with results for 100 runs on the same instance of a maximum satisfiability problem, the benefit of using GRASP instead of repeatedly restarting local search with a randomly generated solution and a greedy solution. Two curves compare objective function value (best and average over the 100 runs) for different values of the RCL parameter α. Two other curves compare solution times (average total time and average local search time) for different values of α. Since this is a maximization problem, $\alpha = 0$ corresponds to random construction, while $\alpha = 1$ corresponds to greedy construction. While the average solution improves as we move from random to greedy, the best solution (what we are really interested in) improves as we move away from random construction, but reaches a maximum before reaching $\alpha = 1$, and then decreases after that. As the mean solution increases, the spread of solutions decreases. The combination of the increase in mean solution value and the presence of enough spread contribute to produce the best solution with $\alpha = .8$. Solution times decrease as one moves from random to greedy and this is mainly due to the decrease in time for local search.

2.1.2 Memory-based mechanisms

If GRASP iteration i uses the random number generator seed s_i, then the iterations are memoryless, i.e. they produce the same result independently of the order in which they are run. In the remainder of this section, we review how the use of memory was introduced into GRASP.

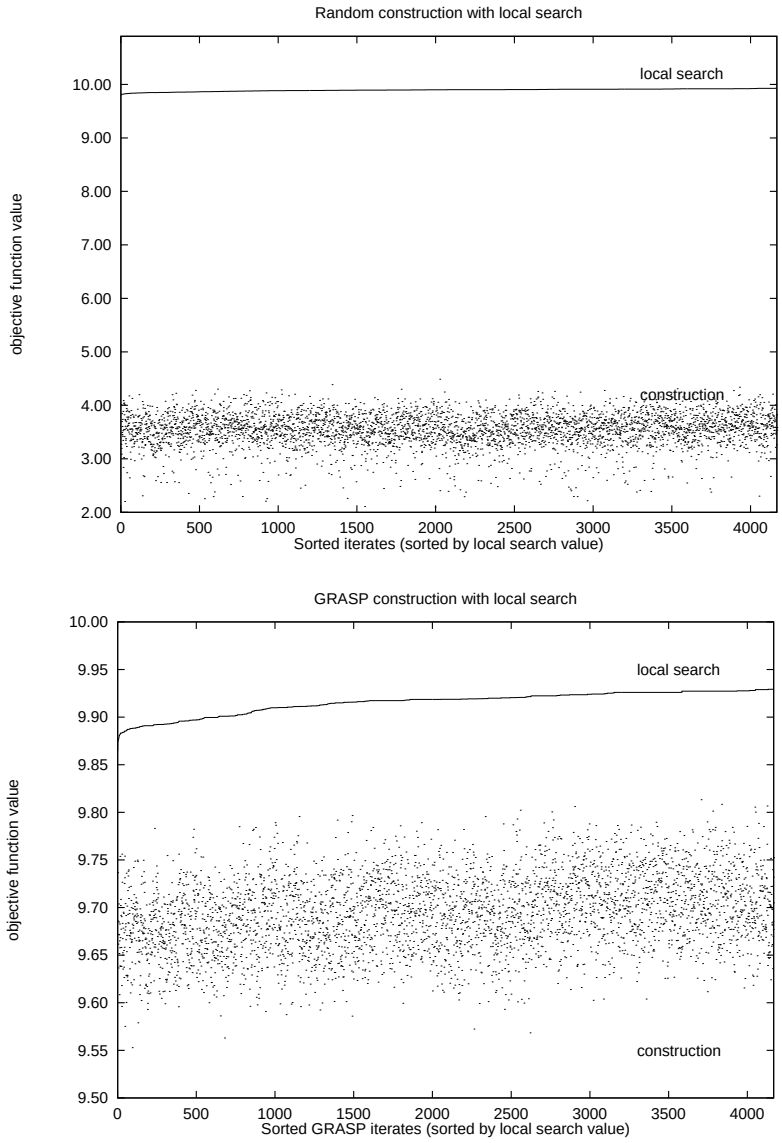


Figure 2.1 Random multi-start vs. GRASP on an instance of maximum covering problem.

Memory can be used to avoid doing redundant work. For example, one can store in a hash table all solutions constructed and used as initial solutions for local search (Martins et al., 1999). Every time a new solution is constructed, it will only be used as an initial solution in the local search phase if it is not present in the hash table.

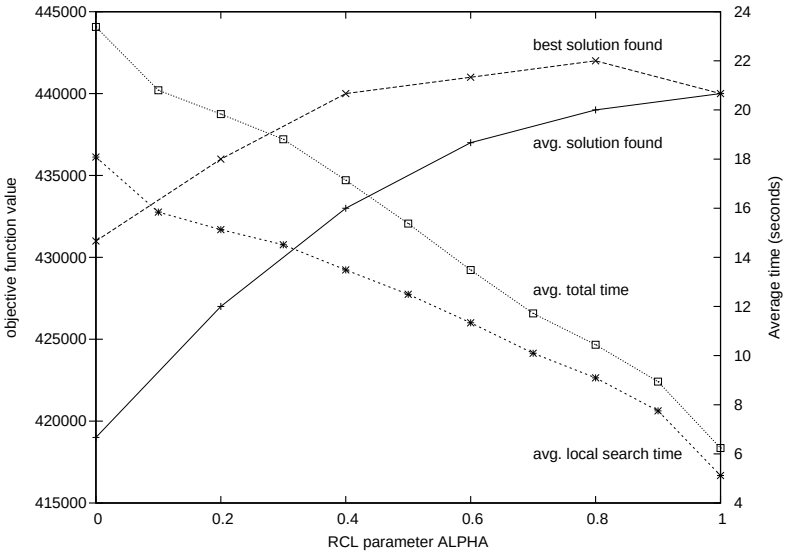


Figure 2.2 Average versus best solution found and total running time versus local search time as a function of the RCL parameter α on 100 runs on an instance of maximum satisfiability.

Filtering of constructed solutions (Feo et al., 1994; Martins et al., 1999; Prais and Ribeiro, 2000) avoids applying local search to low-quality solutions, where local search will probably take long to converge to a low-quality local optimum.

Fleurent and Glover (1999) introduced a long-term memory mechanism in GRASP construction that makes use of a set of elite solutions found during the GRASP iterations. Their mechanism favors (strongly determined) variables that cannot be changed without eroding the objective or changing significantly other variables and (consistent) variables that receive a particular value in a large portion of the elite solutions.

Prais and Ribeiro (2000) introduced another learning mechanism for GRASP construction, which they named *reactive* GRASP. Recall that in a value-based restricted candidate list a parameter α determines the level of randomness or greediness used to make up the RCL. Instead of using a fixed value for α, reactive GRASP selects a value, at random, from a discrete set of values $\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$. Each value α_i has associated with it a probability p_i that it will be selected ($\sum_{i=1}^{m} p_i = 1$). The idea in reactive GRASP is to change these probabilities as the iterations proceed, to favor values that have led to better solutions in previous GRASP iterations.

Laguna and Martí (1999) introduced another strategy for using long-term memory consisting of a set of elite solutions. At each GRASP iteration, this strategy combines the GRASP solution with a randomly selected elite solution, using path-relinking (Glover, 1996). This is the subject of the next section.

2.2 PATH-RELINKING

Path-relinking was originally proposed by Glover (1996) as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search (Glover, 2000; Glover and Laguna, 1997; Glover et al., 2000). Starting from one or more elite solutions, paths in the solution space leading toward other elite solutions are generated and explored in the search for better solutions. To generate paths, moves are selected to introduce attributes in the current solution that are present in the elite guiding solution. Path-relinking may be viewed as a strategy that seeks to incorporate attributes of high quality solutions, by favoring these attributes in the selected moves.

Algorithm 2 illustrates the pseudo-code of the path-relinking procedure applied to a pair of solutions x_s (starting solution) and x_t (target solution).

Data : Starting solution x_s and target solution x_t
Result : Best solution x^* in path from x_s to x_t
Compute symmetric difference $\Delta(x_s, x_t)$;
$f^* \leftarrow \min\{f(x_s), f(x_t)\}$;
$x^* \leftarrow \text{argmin}\{f(x_s), f(x_t)\}$;
$x \leftarrow x_s$;
while $\Delta(x, x_t) \neq \emptyset$ **do**
  $m^* \leftarrow \text{argmin}\{f(x \oplus m) : m \in \Delta(x, x_t)\}$;
  $\Delta(x \oplus m^*, x_t) \leftarrow \Delta(x, x_t) \setminus \{m^*\}$;
  $x \leftarrow x \oplus m^*$;
  if $f(x) < f^*$ **then**
    $f^* \leftarrow f(x)$;
    $x^* \leftarrow x$;
  end
end

Algorithm 2: Path-relinking.

The procedure starts by computing the symmetric difference $\Delta(x_s, x_t)$ between the two solutions, i.e. the set of moves needed to reach x_t (target solution) from x_s (initial solution). A path of solutions is generated linking x_s and x_t. The best solution x^* in this path is returned by the algorithm. At each step, the procedure examines all moves $m \in \Delta(x, x_t)$ from the current solution x and selects the one which results in the least cost solution, i.e. the one which minimizes $f(x \oplus m)$, where $x \oplus m$ is the solution resulting from applying move m to solution x. The best move m^* is made, producing solution $x \oplus m^*$. The set of available moves is updated. If necessary, the best solution x^* is updated. The procedure terminates when x_t is reached, i.e. when $\Delta(x, x_t) = \emptyset$.

We notice that path-relinking may also be viewed as a constrained local search strategy applied to the initial solution x_s, in which only a limited set of moves can be performed and where uphill moves are allowed. Several alternatives have been considered and combined in recent implementations of path-relinking (Aiex, 2002;

Aiex et al., 2003; 2000; Binato et al., 2001; Ribeiro and Rosseti, 2002; Ribeiro et al., 2002; Rosseti, 2003):

- *periodical relinking:* path-relinking is not systematically applied, but instead only periodically;
- *forward relinking:* path-relinking is applied using the worst among x_s and x_t as the initial solution and the other as the target solution;
- *backward relinking:* the roles of x_s and x_t are interchanged, path-relinking is applied using the best among x_s and x_t as the initial solution and the other as the target solution;
- *back and forward relinking:* two different trajectories are explored, the first using x_s as the initial solution and the second using x_t in this role;
- *mixed relinking:* two paths are simultaneously explored, the first emanating from x_s and the second from x_t, until they meet at an intermediary solution equidistant from x_s and x_t;
- *randomized relinking:* instead of selecting the best yet unselected move, randomly select one from among a candidate list with the most promising moves in the path being investigated; and
- *truncated relinking:* the full trajectory between x_s and x_t is not investigated, but instead only part of it.

All these alternatives involve trade-offs between computation time and solution quality. Ribeiro et al. (2002) observed that exploring two different trajectories for each pair (x_s, x_t) takes approximately twice the time needed to explore only one of them, with very marginal improvements in solution quality. They have also observed that if only one trajectory is to be investigated, better solutions are found when the relinking procedure starts from the best among x_s and x_t. Since the neighborhood of the initial solution is much more carefully explored than that of the guiding one, starting from the best of them gives the algorithm a better chance to investigate in more detail the neighborhood of the most promising solution. For the same reason, the best solutions are usually found closer to the initial solution than to the guiding solution, allowing the pruning of the relinking trajectory before the latter is reached.

2.3 GRASP WITH PATH-RELINKING

Path-relinking is a major enhancement to the basic GRASP procedure, leading to significant improvements in solution time and quality.

The use of path-relinking within a GRASP procedure, as an intensification strategy applied to each locally optimal solution, was first proposed by Laguna and Martí (1999). It was followed by several extensions, improvements, and successful applications (Aiex et al., 2000; Canuto et al., 2001; Resende and Ribeiro, 2003c; Resende and Werneck, 2002b; Ribeiro et al., 2002). Two basic strategies are used:

- path-relinking is applied to all pairs of elite solutions, either periodically during the GRASP iterations or after all GRASP iterations have been performed as a post-optimization step; and

- path-relinking is applied as an intensification strategy to each local optimum obtained after the local search phase.

Applying path-relinking as an intensification strategy to each local optimum seems to be more effective than simply using it only as a post-optimization step. In general, combining intensification with post-optimization results in the best strategy. In the context of intensification, path-relinking is applied to pairs (x,y) of solutions, where x is a locally optimal solution produced by each GRASP iteration after local search and y is one of a few elite solutions randomly chosen from a pool with a limited number `Max_Elite` of elite solutions found along the search. Uniform random selection is a simple strategy to implement. Since the symmetric difference is a measure of the length of the path explored during relinking, a strategy biased toward pool elements y with high symmetric difference with respect to x is usually better than one using uniform random selection (Resende and Werneck, 2002b).

The pool is originally empty. Since we wish to maintain a pool of good but diverse solutions, each locally optimal solution obtained by local search is considered as a candidate to be inserted into the pool if it is sufficiently different from every other solution currently in the pool. If the pool already has `Max_Elite` solutions and the candidate is better than the worst of them, then a simple strategy is to have the former replaces the latter. Another strategy, which tends to increase the diversity of the pool, is to replace the pool element most similar to the candidate among all pool elements with cost worse than the candidate's. If the pool is not full, the candidate is simply inserted.

Post-optimization is done on a series of pools. The initial pool P_0 is the pool P obtained at the end of the GRASP iterations. The value of the best solution of P_0 is assigned to f_0^* and the pool counter is initialized $k = 0$. At the k-th iteration, all pairs of elements in pool P_k are combined using path-relinking. Each result of path-relinking is tested for membership in pool P_{k+1} following the same criteria used during the GRASP iterations. If a new best solution is produced, i.e. $f_{k+1}^* < f_k^*$, then $k \leftarrow k+1$ and a new iteration of post-optimization is done. Otherwise, post-optimization halts with $x^* = \text{argmin}\{f(x) \mid x \in P_{k+1}\}$ as the result.

Algorithm 3 illustrates such a procedure. Each GRASP iteration has now three main steps:

- *Construction phase:* a greedy randomized construction procedure is used to build a feasible solution;

- *Local search phase:* the solution built in the first phase is progressively improved by a neighborhood search strategy, until a local minimum is found; and

- *Path-relinking phase:* the path-relinking algorithm using any of the strategies described in Section 2.2 is applied to the solution obtained by local search and to a randomly selected solution from the pool. The best solution found along

this trajectory is also considered as a candidate for insertion in the pool and the incumbent is updated.

At the end of the GRASP iterations, a post-optimization phase combines the elite solutions in the pool in the search for better solutions..

```
Data   : Number of iterations i_max
Result : Solution x* ∈ X
P ← ∅;
f* ← ∞;
for i = 1,...,i_max do
    x ← GreedyRandomizedConstruction();
    x ← LocalSearch(x);
    if i ≥ 2 then
        Choose, at random, pool solutions Y ⊆ P to relink with x;
        for y ∈ Y do
            Determine which (x or y) is initial x_s and which is
            target x_t;
            x_p ← PathRelinking(x_s, x_t);
            Update the elite set P with x_p;
            if f(x_p) < f* then
                f* ← f(x_p);
                x* ← x_p;
            end
        end
    end
end
P = PostOptimize{P};
x* = argmin{f(x), x ∈ P};
```

Algorithm 3: A basic GRASP with path-relinking heuristic for minimization.

Aiex (2002) and Aiex et al. (2002) have shown experimentally that the solution times for finding a target solution value with a GRASP heuristic fit a two-parameter exponential distribution. Figure 2.3 illustrates this result, depicting the superimposed empirical and theoretical distributions observed for one of the cases studied in the computational experiments reported by the authors, which involved 2400 runs of GRASP procedures for each of five different problems: maximum independent set (Feo et al., 1994; Resende et al., 1998), quadratic assignment (Li et al., 1994; Resende et al., 1996), graph planarization (Resende and Ribeiro, 1997; Ribeiro and Resende, 1999), maximum weighted satisfiability (Resende et al., 2000), and maximum covering (Resende, 1998). The same result still holds when GRASP is implemented in conjunction with a path-relinking procedure (Aiex et al., 2003).

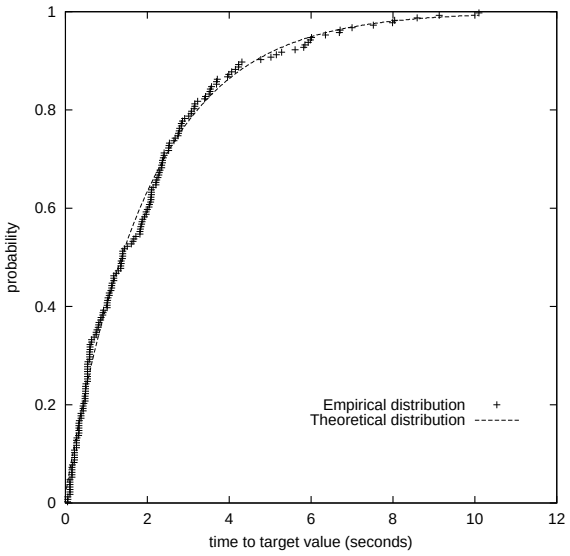


Figure 2.3 Superimposed empirical and theoretical distributions (times to target values measured in seconds on an SGI Challenge computer with 28 processors).

2.4 APPLICATIONS

Path-relinking has been successfully used together with GRASP in a variety of applications, such as the three index assignment problem (Aiex, 2002; Aiex et al., 2000), the problem of routing private circuits in communication networks (Resende and Ribeiro, 2003a), the 2-path network design problem (Ribeiro and Rosseti, 2002), the p-median problem (Resende and Werneck, 2002a), the Steiner problem in graphs (Ribeiro et al., 2002), the job-shop scheduling problem (Aiex, 2002; Aiex et al., 2003), the prize-collecting Steiner tree problem (Canuto et al., 2001), the quadratic assignment problem (Oliveira et al., 2003), the MAX-CUT problem (Festa et al., 2002), and the capacitated minimum spanning tree problem (Souza et al., 2003). Some of these applications will be reviewed in the remainder of this section.

Before we review some of these applications, we first describe a plot used in several of our papers to experimentally compare different randomized algorithms or different versions of the same randomized algorithm (Aiex, 2002; Aiex et al., 2002). This plot shows empirical distributions of the random variable *time to target solution value*. To plot the empirical distribution, we fix a solution target value and run each algorithm T independent times, recording the running time when a solution with cost at least as good as the target value is found. For each algorithm, we associate with the i-th sorted running time (t_i) a probability $p_i = (i - \frac{1}{2})/T$, and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, T$. Figure 2.4 shows one such plot comparing a pure GRASP with a GRASP with path-relinking for MAX-CUT instance G11 with target solution value of 552. The figure shows clearly that GRASP with path-relinking (GRASP+PR) is much faster than pure GRASP to find a solution with weight 552 or more. For instance,

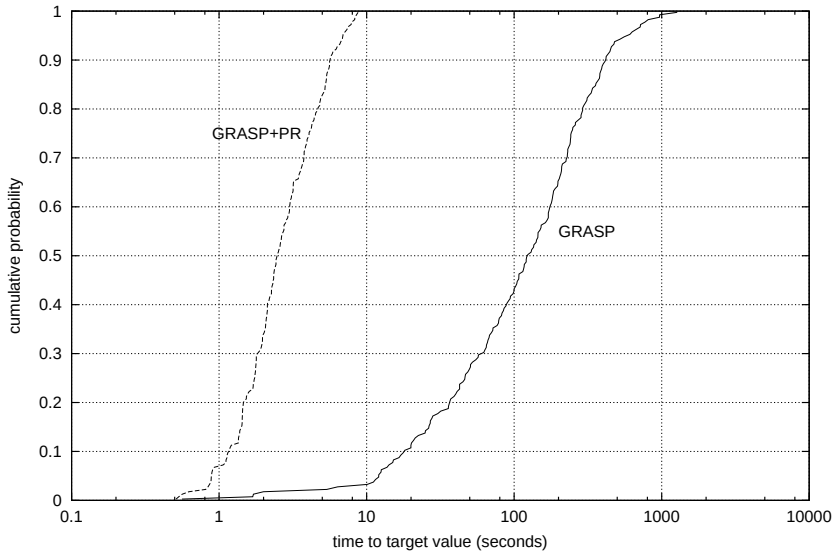


Figure 2.4 Empirical distributions of the random variables *time to target solution* for a pure GRASP and a GRASP with path-relinking for MAX-CUT instance G11 with target value of 552 (Festa et al., 2002). Two hundred independent runs of each algorithm were used to make the plots.

the probability of finding such a solution in less than 5 seconds is over 80% with GRASP with path-relinking, while it is about 2% with pure GRASP. Similarly, with probability 50% GRASP with path-relinking finds such a target solution in less than 2.5 seconds, while for pure GRASP, with probability 50% a solution is found in less than 122 seconds.

2.4.1 Private virtual circuit routing

A frame relay service offers virtual private networks to customers by provisioning a set of long-term private virtual circuits (PVCs) between customer endpoints on a large backbone network. During the provisioning of a PVC, routing decisions are made without any knowledge of future requests. Over time, these decisions can cause inefficiencies in the network and occasional offline rerouting of the PVCs is needed. Resende and Ribeiro (2003c) formulate the offline PVC routing problem as an integer multi-commodity flow problem with additional constraints and with an objective function that minimizes propagation delays and/or network congestion. They propose variants of a GRASP with path-relinking heuristic for this problem. Experimental results for realistic-size problems show that GRASP benefits greatly from path-relinking and that the proposed heuristics are able to improve the solutions found with standard routing techniques.

Let $G = (V, E)$ be an undirected graph representing the frame relay network. Denote by $V = \{1, \ldots, n\}$ the set of backbone nodes where switches reside, while E is

set of trunks (or edges) that connect the backbone nodes, with $|E| = m$. Parallel trunks are allowed. Since G is an undirected graph, flows through each trunk $(i, j) \in E$ have two components to be summed up, one in each direction. However, for modeling purposes, costs and capacities will always be associated only with the ordered pair (i, j) satisfying $i < j$. For each trunk $(i, j) \in E$, denote by b_{ij} its maximum allowed bandwidth (in kbits/second), while c_{ij} denotes the maximum number of PVCs that can be routed through it and d_{ij} is the propagation, or hopping, delay associated with the trunk. Each commodity $k \in K = \{1, \ldots, p\}$ is a PVC to be routed, associated with an origin-destination pair and with a bandwidth requirement (or demand, also known as its effective bandwidth) r_k. The latter takes into account the actual bandwidth required by the customer in the forward and reverse directions, as well as an overbooking factor.

Let $x_{ij}^k = 1$ if and only if edge $(i, j) \in E$ is used to route commodity $k \in K$. The cost function $\phi_{ij}(x_{ij}^1, \ldots, x_{ij}^p, x_{ji}^1, \ldots, x_{ji}^p)$ associated with each trunk $(i, j) \in E$ with $i < j$ is the linear combination of a trunk propagation delay component and a trunk congestion component. The *propagation delay component* is defined as

$$\phi_{ij}^d(x_{ij}^1, \ldots, x_{ij}^p, x_{ji}^1, \ldots, x_{ji}^p) = d_{ij} \cdot \sum_{k \in K} \rho_k (x_{ij}^k + x_{ji}^k), \tag{2.1}$$

where coefficients ρ_k are used to model two plausible delay functions:

- If $\rho_k = 1$, then this component leads to the minimization of the number of hops weighted by the propagation delay on each trunk.
- If $\rho_k = r_k$, then the minimization takes into account the effective bandwidth routed through each trunk weighted by its propagation delay.

Let $y_{ij} = \sum_{k \in K} r_k (x_{ij}^k + x_{ji}^k)$ be the total flow through trunk $(i, j) \in E$ with $i < j$. The *trunk congestion component* depends on the utilization rates $u_{ij} = y_{ij}/b_{ij}$ of each trunk $(i, j) \in E$ with $i < j$. It is taken as the piecewise linear function proposed by Fortz and Thorup (2000) and depicted in Figure 2.5, which increasingly penalizes flows approaching or violating the capacity limits:

$$\phi_{ij}^b(x_{ij}^1, \ldots, x_{ij}^p, x_{ji}^1, \ldots, x_{ji}^p) = \\ = b_{ij} \cdot \begin{cases} u_{ij}, & u_{ij} \in [0, 1/3) \\ 3 \cdot u_{ij} - 2/3, & u_{ij} \in [1/3, 2/3), \\ 10 \cdot u_{ij} - 16/3, & u_{ij} \in [2/3, 9/10), \\ 70 \cdot u_{ij} - 178/3, & u_{ij} \in [9/10, 1), \\ 500 \cdot u_{ij} - 1468/3, & u_{ij} \in [1, 11/10), \\ 5000 \cdot u_{ij} - 16318/3, & u_{ij} \in [11/10, \infty). \end{cases} \tag{2.2}$$

For PVC routing, Resende and Ribeiro used the cost function

$$\phi_{ij}(x_{ij}^1, \ldots, x_{ij}^p, x_{ji}^1, \ldots, x_{ji}^p) = \\ = (1 - \delta) \cdot \phi_{ij}^d(x_{ij}^1, \ldots, x_{ij}^p, x_{ji}^1, \ldots, x_{ji}^p) + \delta \cdot \phi_{ij}^b(x_{ij}^1, \ldots, x_{ij}^p, x_{ji}^1, \ldots, x_{ji}^p) \tag{2.3}$$

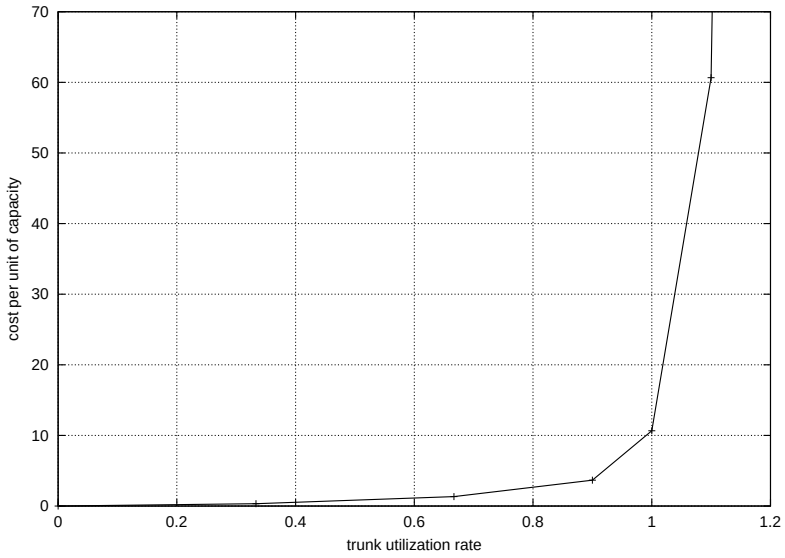


Figure 2.5 Piecewise linear congestion cost component associated with each trunk.

associated with each trunk $(i, j) \in E$ with $i < j$, where weights $(1-\delta)$ and δ correspond respectively to the propagation delay and the network congestion components, with $\delta \in [0, 1]$.

In the construction phase of GRASP, the routes are determined, one at a time. A new PVC is selected to be routed in each iteration. To reduce the computation times, we used a combination of the strategies usually employed by GRASP and heuristic-biased stochastic sampling. We create a restricted candidate list with a fixed number of elements n_c. At each iteration, it is formed by the n_c unrouted PVC pairs with the largest demands. An element ℓ is selected at random from this list with probability $\pi(\ell) = r_\ell / \sum_{k \in \mathrm{RCL}} r_k$.

Once a PVC $\ell \in K$ is selected, it is routed on a shortest path from its origin to its destination. The bandwidth capacity constraints are relaxed and handled via the penalty function introduced by the trunk congestion component (2.2) of the edge weights. The constraints on the limit of PVCs routed through each trunk are explicitly taken into account by forbidding routing through trunks already using its maximum number of PVCs. The weight $\Delta\phi_{ij}$ of each edge $(i, j) \in E$ is given by the increment of the cost function value $\phi_{ij}(x^1_{ij}, \ldots, x^p_{ij}, x^1_{ji}, \ldots, x^p_{ji})$, associated with routing r_ℓ additional units of demand through edge (i, j).

More precisely, let $\underline{K} \subseteq K$ be the set of previously routed PVCs and $\underline{K}_{ij} \subseteq \underline{K}$ be the subset of PVCs that are routed through trunk $(i, j) \in E$. Likewise, let $\overline{K} = \underline{K} \cup \{\ell\} \subseteq K$ be the new set of routed PVCs and $\overline{K}_{ij} = \underline{K}_{ij} \cup \{\ell\} \subseteq \overline{K}$ be the new subset of PVCs that are routed through trunk (i, j). Then, define $\underline{x}^h_{ij} = 1$ if PVC $h \in \underline{K}$ is routed through trunk $(i, j) \in E$ from i to j, $\underline{x}^h_{ij} = 0$ otherwise. Similarly, define $\overline{x}^h_{ij} = 1$ if PVC $h \in \overline{K}$ is routed through trunk $(i, j) \in E$ from i to j, $\overline{x}^h_{ij} = 0$ otherwise. According

to (2.3), the cost associated with each edge $(i, j) \in E$ in the current solution is given by $\phi_{ij}(\underline{x}^1_{ij}, \ldots, \underline{x}^p_{ij}, \underline{x}^1_{ji}, \ldots, \underline{x}^p_{ji})$. In the same manner, the cost associated with each edge $(i, j) \in E$ after routing PVC ℓ will be $\phi_{ij}(\overline{x}^1_{ij}, \ldots, \overline{x}^p_{ij}, \overline{x}^1_{ji}, \ldots, \overline{x}^p_{ji})$. Then, the incremental edge weight $\Delta\phi_{ij}$ associated with routing PVC $\ell \in K$ through edge $(i, j) \in E$, used in the shortest path computations, is given by

$$\Delta\phi_{ij} = \phi_{ij}(\overline{x}^1_{ij}, \ldots, \overline{x}^p_{ij}, \overline{x}^1_{ji}, \ldots, \overline{x}^p_{ji}) - \phi_{ij}(\underline{x}^1_{ij}, \ldots, \underline{x}^p_{ij}, \underline{x}^1_{ji}, \ldots, \underline{x}^p_{ji}). \tag{2.4}$$

The enforcement of the constraints that limit the number of PVCs routed through each trunk may lead to unroutable demand pairs. In this case, the current solution is discarded and a new construction phase starts.

Each solution built in the first phase may be viewed as a set of routes, one for each PVC. The local search procedure seeks to improve each route in the current solution. For each PVC $k \in K$, start by removing r_k units of flow from each edge in its current route. Next, compute incremental edge weights $\Delta\phi_{ij}$ associated with routing this demand through each trunk $(i, j) \in E$ according to (2.4). A tentative new shortest path route is computed using the incremental edge weights. If the new route improves the solution, it replaces the current route of PVC k. This is continued until no improving route can be found.

In the proposed path-relinking strategy, the set of moves corresponding to the symmetric difference $\Delta(x_1, x_2)$ between any pair $\{x_1, x_2\}$ of solutions is the subset $K_{x_1,x_2} \subseteq K$ of PVCs routed through different routes in x_1 and x_2. Without loss of generality, suppose that path-relinking starts from any elite solution z in the pool and uses the locally optimal solution y as the guiding solution.

The best solution $\overline{y}$ along the new path to be constructed is initialized with z. For each PVC $k \in K_{y,z}$, the same shortest path computations described for construction and local search are used to evaluate the cost of the new solution obtained by rerouting the demand associated with PVC k through the route used in the guiding solution y instead of that used in the current solution originated from z. The best move is selected and removed from $K_{y,z}$. The new solution obtained by rerouting the above selected PVC is computed, the incumbent $\overline{y}$ is updated, and a new iteration resumes. These steps are repeated, until the guiding solution y is reached. The incumbent $\overline{y}$ is returned as the best solution found by path-relinking and inserted into the pool if it is better than the worst solution currently in the pool.

Figure 2.6 illustrates the comparison of the four algorithms: pure GRASP (`GRASP`), GRASP with forward path-relinking (`GRASP+PRf`, in which a locally optimal solution is used as the initial solution), GRASP with backward path-relinking (`GRASP+PRb`, in which an elite solution is used as the initial solution), and GRASP with backward and forward path-relinking (`GRASP+PRfb`, in which path-relinking is performed in both directions) on PVC routing instance `fr750a` (60 nodes, 498 arcs, and 750 commodities). For a given computation time, the probability of finding a solution at least as good as the target value increases from `GRASP` to `GRASP+PRf`, from `GRASP+PRf` to `GRASP+PRfb`, and from `GRASP+PRfb` to `GRASP+PRb`. For example, there is 9.25% probability for `GRASP+PRfb` to find a target solution in less than 100 seconds, while this probability increases to 28.75% for `GRASP+PRb`. For `GRASP`, there is a 8.33% probability of finding a target solution within 2000 seconds, while for `GRASP+PRf` this probability increases

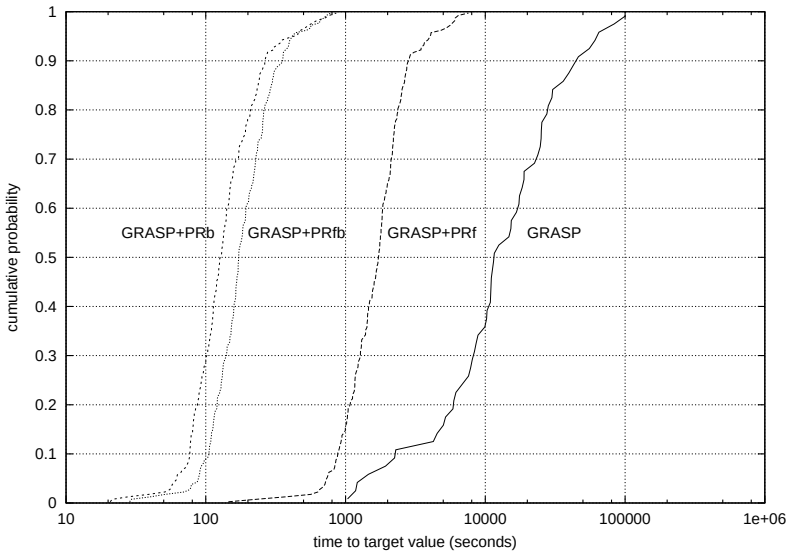


Figure 2.6 Empirical distributions of time to target solution for GRASP, GRASP with forward path-relinking, GRASP with backward path-relinking, and GRASP with back and forward path-relinking for private virtual circuit routing instance `fr750a`. Two hundred independent runs were done for each algorithm. Target solution value used was 479000.

to 65.25%. `GRASP+PRb` finds a target solution in at most 129 seconds with 50% probability. For the same probability, this time increases to 172, 1727, and 10933 seconds, respectively, for variants `GRASP+PRfb`, `GRASP+PRf`, and `GRASP`.

These results suggest that variant `GRASP+PRb`, which performs path-relinking backward from an elite solution to a locally optimal solution, is the most effective.

Another experiment comparing the four variants was done on PVC routing instance `att` (90 nodes, 274 trunks, 272 commodities). Ten independent runs of each algorithm were done for 100 seconds on a 196 MHz SGI Challenge computer. Table 2.1 summarizes these results. For each variant, this table lists the best and average solution values found after 10 seconds and after 100 seconds. The results point to `GRASP+PRb` and `GRASP+PRfb` as the two best heuristics. It is interesting to note that even if given 100 seconds, `GRASP` finds solutions of worse quality than those found by `GRASP+PRb` and `GRASP+PRfb` in only 10 seconds.

2.4.2 2-path network design

Let $G = (V,E)$ be a connected undirected graph, where V is the set of nodes and E is the set of edges. A k-path between nodes $s,t \in V$ is a sequence of at most k edges connecting them. Given a non-negative weight function $w : E \rightarrow R_+$ associated with the edges of G and a set D of pairs of origin-destination nodes, the *2-path network design problem* (2PNDP) consists in finding a minimum weighted subset of edges $E' \subseteq E$ containing a 2-path between every origin-destination pair. Applications can

Table 2.1 Comparison of GRASP, GRASP with forward path-relinking, GRASP with backward path-relinking, and GRASP with back and forward path-relinking for private virtual circuit routing instance `att`. Ten independent runs of 100 seconds were done for each algorithm.

10 runs	10 seconds		100 seconds	
Variant	best	average	best	average
`GRASP`	126603	126695	126228	126558
`GRASP+PRf`	126301	126578	126083	126229
`GRASP+PRb`	125960	126281	125666	125883
`GRASP+PRfb`	125961	126307	125646	125850

be found in the design of communication networks, in which paths with few edges are sought to enforce high reliability and small delays. Dahl and Johannessen (2000) proved that the decision version of 2PNDP is NP-complete.

Rosseti (2003) and Ribeiro and Rosseti (2002) described sequential and parallel implementations of GRASP with path relinking for the 2-path network design. The construction of a new solution begins by the initialization of modified edge weights with the original edge weights. Each iteration of the construction phase starts by the random selection of an origin-destination pair still in D. A shortest 2-path between the extremities of this pair is computed, using the modified edge weights. The weights of the edges in this 2-path are set to zero until the end of the construction procedure, the origin-destination pair is removed from D, and a new iteration resumes. The construction phase stops when 2-paths have been computed for all origin-destination pairs.

The local search phase seeks to improve each solution built in the construction phase. Each solution may be viewed as a set of 2-paths, one for each origin-destination pair in D. To introduce some diversity by driving different applications of the local search to different local optima, the origin-destination pairs are investigated at each GRASP iteration in a circular order defined by a different random permutation of their original indices. Each 2-path in the current solution is tentatively eliminated. The weights of the edges used by other 2-paths are temporarily set to zero, while those which are not used by other 2-paths in the current solution are restored to their original values. A new shortest 2-path between the extremities of the origin-destination pair under investigation is computed, using the modified weights. If the new 2-path improves the current solution, then the latter is modified; otherwise the previous 2-path is restored. The search stops if the current solution was not improved after a sequence of $|D|$ iterations along which all 2-paths have been investigated. Otherwise, the next 2-path in the current solution is investigated for substitution and a new iteration resumes.

Each GRASP iteration performs an intensification phase using path-relinking, in which the newly generated solution obtained at the end of the local search phase is combined with a randomly selected solution from the pool of elite solutions. Path-relinking starts by determining all origin-destination pairs whose associated 2-paths

are different in the two solutions. These computations amount to determining the set of moves which should be applied to the initial solution to reach the guiding one. Each move is characterized by a pair of 2-paths, one to be inserted and the other to be eliminated from the current solution. At each path-relinking iteration the best yet unselected move is applied to the current solution and the best solution found along the path connecting the two solutions is updated. The incumbent best solution found along the path-relinking step is inserted into the pool if it is better than the worst solution currently in the pool.

Several strategies for the implementation of the path-relinking step have been investigated in Ribeiro and Rosseti (2004, in preparation); Rosseti (2003): pure GRASP (`GRASP`), GRASP with forward path-relinking (`GRASP+PRf`, in which a locally optimal solution is used as the initial solution), GRASP with backward path-relinking (`GRASP+PRb`, in which an elite solution is used as the initial solution), GRASP with backward and forward path-relinking (`GRASP+PRfb`, in which path-relinking is performed twice, once in each direction), and GRASP with mixed path-relinking (`GRASP+PRm`, in which two paths in opposite directions are simultaneously explored).

The results displayed in Table 2.2 illustrate the behavior of these five variants on randomly generated instances (Rosseti, 2003) on complete graphs with 100, 200, 300, 400, and 500 nodes. For each instance, we give the best and average solution values found over ten independent runs of each algorithm. For each problem size, the processing time is limited at that observed for 200 iterations of the pure GRASP procedure on the first instance in the group. Algorithms `GRASP+PRfb` and `GRASP+PRm` performed better than the other variants, as far as together they found the best solutions and the best average solutions for all instances in the table. GRASP with backward path-relinking usually performs better than the forward path-relinking variant, due to the fact that it starts from an elite solution that is often better than the current local optimum, fully exploring the neighborhood of the former.

The results observed for variant `GRASP+PRm` are very encouraging: this algorithm found better solutions than the other variants for 40% of the instances.

To further illustrate and compare these five variants, we display in Figure 2.7 a plot of the empirical probability distribution of the time to target solution value for each algorithm, computed from 200 independent runs. These plots show that the probability of finding a solution at least as good as a target value increases from `GRASP` to `GRASP+PRf` to `GRASP+PRb` to `GRASP+PRfb`, and finally to `GRASP+PRm`. These results confirms an observation first noticed by Ribeiro et al. (2002) and later by Resende and Ribeiro (2003b), suggesting that the backward strategy performs a major role in successful implementations of path-relinking. Moreover, they also indicate that the mixed path-relinking strategy proposed by Rosseti (2003) is very effective.

2.4.3 ***p****-median problem*

In the p-median problem, we are given a set F of m potential facilities, a set U of n users (or customers), a distance function $d : U \times F \rightarrow \mathbb{R}$, and a constant $p \leq m$, and want to determine which p facilities to open so as to minimize the sum of the distances from each user to its closest open facility. It is a well-known NP-hard problem (Kariv and Hakimi, 1979).

Table 2.2 Results for ten runs of each algorithm on randomly generated instances of 2-path network design problems with limited processing time.

	GRASP		GRASP+PRf		GRASP+PRb		GRASP+PRfb		GRASP+PRm	
$\|V\|$	best	avg.	best	avg.	best	avg.	best	avg.	best	avg.
100	779	784.3	760	772.8	763	769.3	749	762.7	755	765.3
	762	769.6	730	749.4	735	746.0	729	741.7	736	745.7
	773	779.2	762	769.3	756	766.1	757	763.6	754	765.1
	746	752.0	732	738.4	723	736.7	719	730.4	717	732.2
	756	762.3	742	749.7	739	746.5	737	742.9	728	743.7
200	1606	1614.7	1571	1584.4	1540	1568.0	1526	1562.0	1538	1564.3
	1601	1608.8	1557	1572.8	1559	1567.9	1537	1558.9	1545	1563.3
	1564	1578.2	1523	1541.9	1516	1531.9	1508	1519.9	1509	1528.7
	1578	1585.6	1531	1553.3	1518	1538.1	1510	1532.2	1513	1534.7
	1577	1599.6	1567	1575.4	1543	1563.5	1529	1556.3	1531	1556.1
300	2459	2481.9	2408	2425.0	2377	2401.3	2355	2399.2	2366	2393.6
	2520	2527.7	2453	2469.7	2419	2449.1	2413	2438.9	2405	2439.4
	2448	2463.5	2381	2403.1	2339	2373.8	2356	2375.3	2338	2370.3
	2462	2482.1	2413	2436.2	2373	2409.3	2369	2400.9	2350	2401.0
	2450	2458.8	2364	2402.5	2328	2368.6	2347	2373.9	2322	2365.4
400	3355	3363.8	3267	3285.5	3238	3257.0	3221	3239.4	3231	3252.2
	3393	3417.5	3324	3338.2	3283	3306.8	3220	3292.2	3271	3301.4
	3388	3394.4	3311	3322.4	3268	3291.9	3227	3275.1	3257	3273.2
	3396	3406.0	3316	3326.5	3249	3292.0	3256	3284.8	3246	3287.9
	3416	3429.3	3335	3365.5	3267	3327.7	3270	3313.9	3259	3323.5
500	4338	4350.1	4209	4247.1	4176	4207.6	4152	4196.1	4175	4206.2
	4353	4369.6	4261	4278.6	4180	4233.7	4166	4219.6	4175	4226.3
	4347	4360.7	4239	4257.8	4187	4224.8	4170	4201.9	4187	4217.9
	4317	4333.8	4222	4238.6	4157	4197.4	4156	4182.2	4159	4197.1
	4362	4370.4	4263	4292.0	4203	4294.0	4211	4236.8	4200	4240.2

Resende and Werneck (2002b) describe a GRASP with path-relinking for the p-median problem. Empirical results on instances from the literature show that the algorithm is robust and that it performs at least as well as other methods, and often better in terms of both running time and solution quality. In all cases the solutions obtained by the GRASP with path-relinking were within 0.1% of the best known upper bounds. For a large number of instances new best known solutions were produced by the new algorithm.

The standard greedy algorithm for the p-median problem (Cornuejols et al., 1977; Whitaker, 1983) starts with an empty solution and adds facilities one at a time, choosing the most profitable in each iteration (the one whose insertion causes the greatest drop in solution cost). The construction procedure proposed in Resende and Werneck (2002b) is similar to the greedy algorithm, but instead of selecting the best among all possible options, it only considers $q < m$ possible insertions (chosen uniformly at ran-

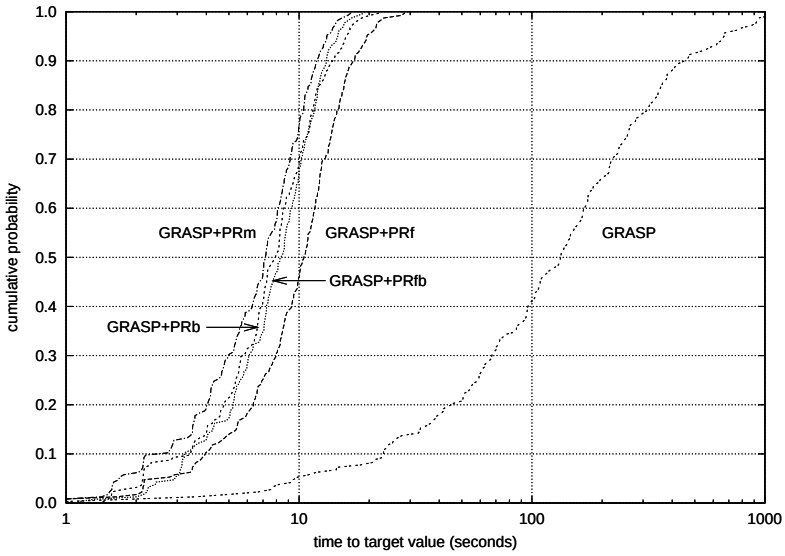


Figure 2.7 Empirical distributions of time to target solution for GRASP, GRASP with forward path-relinking, GRASP with backward path-relinking, GRASP with back and forward path-relinking, and GRASP with mixed path-relinking for a 2-path network design instance with 80 nodes. Two hundred independent runs were done for each algorithm. Target solution value used was 588.

dom) in each iteration. The most profitable among those is selected. The running time of the algorithm is $O(m+pqn)$. The idea is to make q small enough so as to significantly reduce the running time of the algorithm (when compared to the pure greedy one) and to ensure a fair degree of randomization. In tests, the value $q = \lceil \log_2(m/p) \rceil$ was determined to be suitable.

The standard local search procedure for the p-median problem, originally proposed by Teitz and Bart (1968), is based on swapping facilities. Given an initial solution S, the procedure determines, for each facility $f \notin S$, which facility $g \in S$ (if any) would improve the solution the most if f and g were interchanged (i.e., if f were opened and g closed). If there is one such improving move, f and g are interchanged. The procedure continues until no improving interchange can be made, in which case a local minimum will have been found. The complexity of this swap-based local search is $O(pmn)$ per iteration. Whitaker (1983) proposed an efficient implementation of this method, which he called *fast interchange*, for which the bound on the running time of each iteration is reduced to $O(mn)$. Resende and Werneck (2003a) have recently proposed an alternative implementation. Although it has the same worst-case complexity as Whitaker's, it can be substantially faster in practice. The speedup (of up to three orders of magnitude) results from the use of information gathered in early iterations of the algorithm to reduce the amount of computation performed in later stages. Though this implementation can require a greater amount of memory, with the use of some

programming techniques (e.g. sparse matrix representation and cache), the additional memory requirements can be minimized.

Intensification (via path-relinking) occurs in two different stages. First, every GRASP iteration contains an intensification step, in which the newly generated solution is combined with a solution from the pool. Then, in the post-optimization phase, solutions in the pool are combined among themselves.

Let S_1 and S_2 be two valid solutions, interpreted as sets of (open) facilities. The path-relinking procedure starts with one of the solutions (say, S_1) and gradually transforms it into the other (S_2) by swapping in elements from $S_2 \setminus S_1$ and swapping out elements from $S_1 \setminus S_2$. The total number of swaps made is $|S_2 \setminus S_1|$, which is equal to $|S_1 \setminus S_2|$. The choice of which swap to make in each stage is greedy: the most profitable (or least costly) move is made.

The outcome of path-relinking is the best local minimum in the path. A local minimum in this context is a solution that is both succeeded (immediately) and preceded (either immediately or through a series of same-value solutions) in the path by strictly worse solutions. If the path has no local minima, one of the original solutions (S_1 or S_2) is returned with equal probability. When there is an improving solution in the path, this criterion matches the traditional one exactly: it simply returns the best element in the path. It is different only when the standard path-relinking is unsuccessful, in which case it tries to increase diversity by selecting a solution other than the extremes of the path.

Note that path-relinking is very similar to the local search procedure described earlier, with two main differences. First, the number of allowed moves is restricted: only elements in $S_2 \setminus S_1$ can be inserted, and only those in $S_1 \setminus S_2$ can be removed. Second, non-improving moves are allowed. These differences are easily incorporated into the basic implementation of the local search procedure.

The intensification procedure is augmented by performing a full local search on the solution produced by path-relinking. Because this solution is usually very close to a local optimum, this application tends to be much faster than on a solution generated by the randomized constructive algorithm. A side effect of applying local search at this point is increased diversity, since one is free to use facilities that did not belong to any of the original solutions.

The plots in Figure 2.8 compare GRASP with path-relinking and pure GRASP on the 1400-facility, 1400-user TSPLIB instance fl1400. The plot on the left shows quality of the best solution found as a fraction of the average value of the first solution for GRASP with path-relinking and pure GRASP for $p = 500$. Times are given as multiples of the average time required to perform one multi-start iteration. Smaller values are better. The plot on the right shows ratios between partial solutions found with and without path-relinking for different values of p. Ratios smaller than 1.000 favor the use of path-relinking. The plots show that GRASP benefits from path-relinking, in particular for large values of p.

2.4.4 Three index assignment problem

The three-index assignment problem (AP3) was introduced by Pierskalla (1967) as an extension of the classical two-dimensional assignment problem. Consider a complete

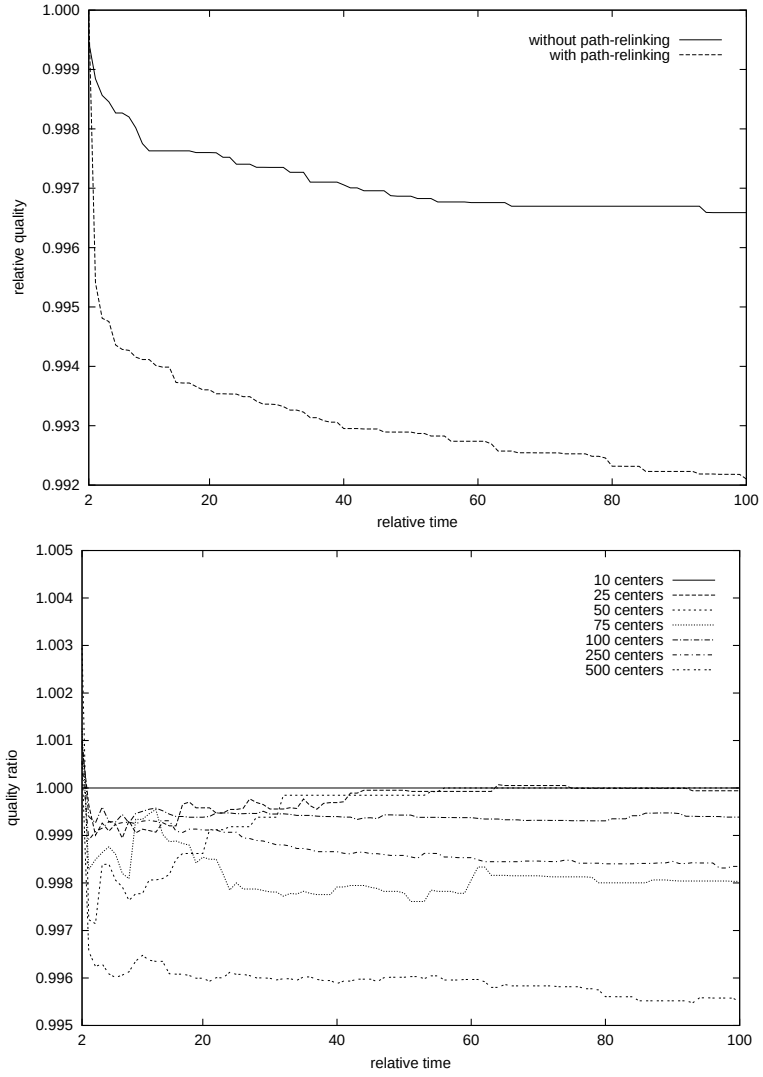


Figure 2.8 GRASP with path-relinking versus pure GRASP on TSPLIB instance fl1400.

tripartite graph $K_{n,n,n} = (I \cup J \cup K, (I \times J) \cup (I \times K) \cup (J \times K))$,where I, J, and K are disjoint sets of size n. If a cost $c_{i,j,k}$ is associated with each triplet $(i, j, k) \in I \times J \times K$, then the AP3 consists of finding a subset $A \in I \times J \times K$ of n triplets such that every element of $I \cup J \cup K$ occurs in exactly one triplet of A, and the sum of the costs of the chosen triplets is minimized. The AP3 is NP-hard (Frieze, 1983; Garey and Johnson,

1979). A permutation-based formulation for AP3 is

$$\min_{p,q\in\pi_N} \sum_{i=1}^{n} c_{ip(i)q(i)},$$

where π_N denotes the set of all permutations of the set of integers $N = \{1,2,\ldots,n\}$.

Aiex (2002) and Aiex et al. (2000) describe a GRASP with path-relinking for AP3. Computational results show clearly that this GRASP for AP3 benefits from path-relinking and compares well with previously proposed heuristics for this problem. GRASP with path-relinking was able to improve the solution quality of heuristics proposed by Balas and Saltzman (1991), Burkard et al. (1996), and Crama and Spieksma (1992) on all instances proposed in those papers.

The GRASP construction phase builds a feasible solution S by selecting n triplets, one at a time. The solution S is initially empty and the set C of candidate triplets is initially the set of all triplets. To select the pth triplet ($p = 1,\ldots,n-1$) to be added to the solution, a restricted candidate list C' is defined to include all triplets (i,j,k) in the candidate set C having cost $c_{ijk} \leq \underline{c} + \alpha(\overline{c} - \underline{c})$, where

$$\underline{c} = \min\{c_{ijk} \mid (i,j,k) \in C\} \text{ and } \overline{c} = \max\{c_{ijk} \mid (i,j,k) \in C\}.$$

Triplet $(i_p, j_p, k_p) \in C'$ is chosen at random and is added to the solution, i.e. $S = S \cup \{(i_p, j_p, k_p)\}$. Once (i_p, j_p, k_p) is selected, the set of candidate triplets must be adjusted to take into account that (i_p, j_p, k_p) is part of the solution. Any triplet (i,j,k) such that $i = i_p$ or $j = j_p$ or $k = k_p$ is removed from the current set of candidate triplets. After $n-1$ triplets have been selected, the set C of candidate triplets contains one last triplet which is added to S, thus completing the construction phase.

In the local search procedure, the current solution is improved by searching its neighborhood for a better solution. The solution of the AP3 can be represented by a pair of permutations (p,q). For a solution $p,q \in \pi_N$, the 2-exchange neighborhood is $N_2(p,q) = \{p',q' \mid d(p,p') + d(q,q') = 2\}$, where $d(s,s') = |\{i \mid s(i) \neq s'(i)\}|$.

In the local search, each cost of a neighborhood solution is compared with the cost of the current solution. If the cost of the neighbor is lower, then the solution is updated, the search is halted, and a search in the new neighborhood is initialized. The local search ends when no neighbor of the current solution has a lower cost than the current solution.

Path-relinking is done between an initial solution

$$S = \{(1, j_1^S, k_1^S), (2, j_2^S, k_2^S), \ldots, (n, j_n^S, k_n^S)\}$$

and a guiding solution

$$T = \{(1, j_1^T, k_1^T), (2, j_2^T, k_2^T), \ldots, (n, j_n^T, k_n^T)\}.$$

Let the symmetric difference between S and T be defined by the following two sets of indices:

$$\delta J = \{i = 1,\ldots,n \mid j_i^S \neq j_i^T\}$$

and

$$\delta K = \{i = 1,\ldots,n \mid k_i^S \neq k_i^T\}.$$

An intermediate solution of the path is visited at each step of path-relinking. Two elementary types of moves can be carried out. In a type-one move, triplets

$$\{(i_1, j_1, k_1), (i_2, j_2, k_2)\}$$

are replaced by triplets

$$\{(i_1, j_2, k_1), (i_2, j_1, k_2)\}.$$

In a type-two move, triplets

$$\{(i_1, j_1, k_1), (i_2, j_2, k_2)\}$$

are replaced by

$$\{(i_1, j_1, k_2), (i_2, j_2, k_1)\}.$$

Set δJ guides type-one moves, while δK guides type-two moves. For all $i \in \delta J$, let q be such that $j_q^T = j_i^S$. A type-one move replaces triplets

$$\{(i, j_i^S, k_i^S), (q, j_q^S, k_q^S)\}$$

by

$$\{(i, j_q^S, k_i^S), (q, j_i^S, k_q^S)\}.$$

For all $i \in \delta K$, let q be such that $k_q^T = k_i^S$. A type-two move replaces triplets

$$\{(i, j_i^S, k_i^S), (q, j_q^S, k_q^S)\}$$

by

$$\{(i, j_i^S, k_q^S), (q, j_q^S, k_i^S)\}.$$

At each step, the move that produces the least costly solution is selected and the corresponding index is deleted from either δJ or δK. This process continues until there are only two move indices left in one of the sets δJ or δK. At this stage, any of these two moves results in the *guiding solution* and, therefore, are not carried out. The best solution found in the path is returned by the procedure.

The plots in Figure 2.9 illustrate how GRASP with path-relinking compares with pure GRASP and how different variants of GRASP with path-relinking compare. The variants of GRASP with path-relinking tested were: random selection of one guiding solution [GPR(RAND)]; random selection of one guiding solution and periodic relinking of all elements in pool [GPR(RAND,INT)]; selection of all pool elements as guiding solutions [GPR(ALL)]; and selection of all pool elements as guiding solutions with periodic relinking of all elements in pool [GPR(ALL,INT)]. The algorithms were run 200 times (using different initial seeds for the random number generator) on instance 24.1 of Balas and Saltzman (1991), stopping when a solution value better than a given target value was found. The experiment comparing pure GRASP with GRASP with path-relinking used a target value of 17, while the one comparing the different variants of GRASP with path-relinking used a more difficult target value of 7. The plot on the left shows the benefit of using path-relinking in GRASP. The plot on the right shows that the variants using path-relinking with all elite solutions have a higher probability of finding a target solution in a given amount of time than the variants that use path-relinking with a single randomly selected elite solution. The use of periodic intensification does not appear to influence the distributions as much.

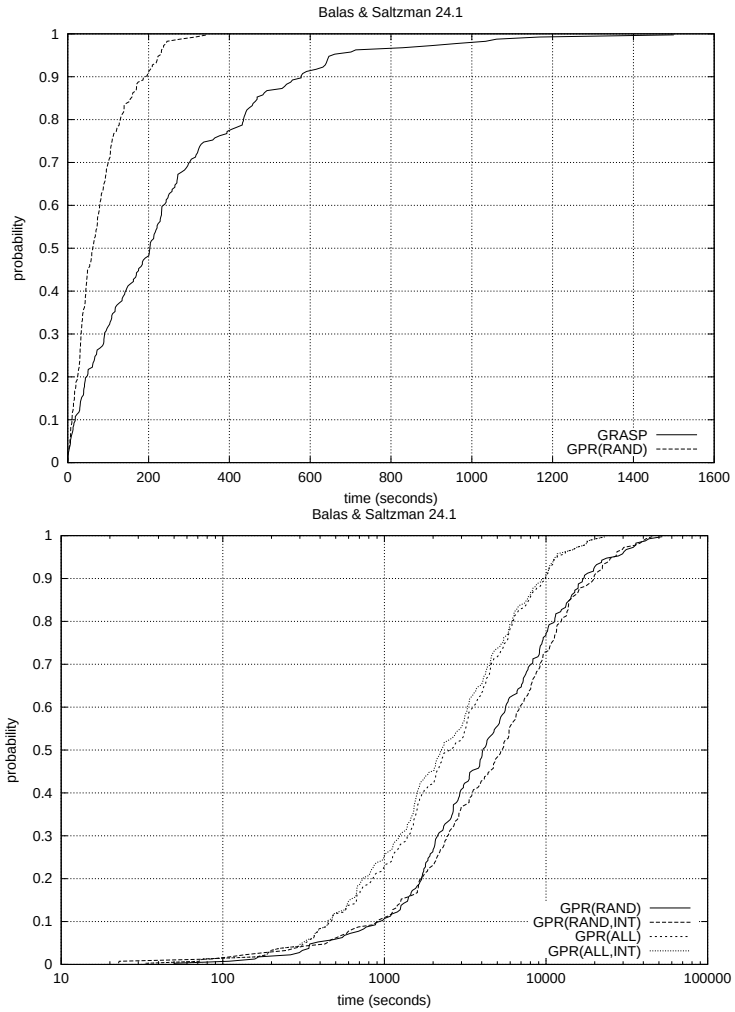


Figure 2.9 The plot of the left shows empirical probability distributions of time to target value for GRASP and GRASP with path-relinking (random selection of one guiding solution from elite set). The plot of the right shows empirical probability distributions of time to target value for different variants of GRASP with path-relinking.

2.5 CONCLUSIONS AND EXTENSIONS

This paper reviewed recent advances and applications of GRASP with path-relinking. By providing a short discussion of each component of GRASP with path-relinking and showing examples of how such heuristics can be implemented for combinatorial optimization problems such as PVC routing, 2-path network design, 3-index assignment,

and p-median, we hope this paper will serve as a guide for the reader to put together other GRASP with path-relinking heuristics.

Path-relinking is a major enhancement to the basic greedy randomized adaptive search procedure (GRASP), leading to significant improvements in both solution time and quality. It adds an effective memory mechanism to GRASP by providing an intensification strategy that explores trajectories connecting GRASP solutions and the best elite solutions previously produced during the search. The numerical results summarized for the four problems listed above clearly illustrate the benefits obtained by the combination of GRASP with path relinking.

In *evolutionary path-relinking* used in the post-optimization intensification phase, a new generation of elite solutions is generated from the current population in the pool of elite solutions by applying path-relinking between all pairs of solutions in this population. Solutions obtained by each path-relinking operation are tested for inclusion in the population of the next generation following the usual rules used in pool management. This strategy was successfully used for the Steiner problem in graphs by Ribeiro et al. (2002), for the p-median problem by Resende and Werneck (2002a), and for the uncapacitated facility location problem by Resende and Werneck (2003b).

Path-relinking may also be used as a solution extractor for population methods. In particular, path-relinking was recently successfully applied as a generalized crossover strategy to generate optimized offsprings in the context of a genetic algorithm for the phylogeny problem (Ribeiro and Vianna, 2003).

The fact that the computation time to find a target solution value using GRASP with path-relinking fits a two-parameter exponential distribution (cf. Section 2.3, see (Aiex, 2002; Aiex et al., 2003; 2002)) has a major consequence in parallel implementations of GRASP with path-relinking: linear speedups proportional to the number of processors can be easily observed in *parallel independent strategies*. Additionally, path-relinking offers a very effective mechanism for the implementation of *parallel cooperative strategies* (Cung et al., 2002). In this case, inter-processor cooperation is enforced by a master processor which stores and handles a common pool of elite solutions which is shared by all slave processors performing GRASP with path-relinking. Careful implementations making appropriate use of the computer resources may lead to even larger speedups and to very robust parallel algorithms, see e.g. (Martins et al., 2004; Ribeiro and Rosseti, 2002; 2004, in preparation; Rosseti, 2003). Results obtained for the 2-path network design problem are illustrated in Figure 2.10, showing the speedup obtained by the cooperative strategy with respect to the independent one on a cluster of eight processors. Much larger improvements can be obtained with more processors.

Finally, we notice that path-relinking can also be successfully used in conjunction with implementations of other metaheuristics such as VNS and ant colonies, as recently reported e.g. in Aloise et al. (2003); Festa et al. (2002).

Acknowledgments. Most of this work is part of their dissertations and was jointly done with the following currently and former M.Sc. and Ph.D. students from the Catholic University of Rio de Janeiro, Brazil, which are all gratefully acknowledged:

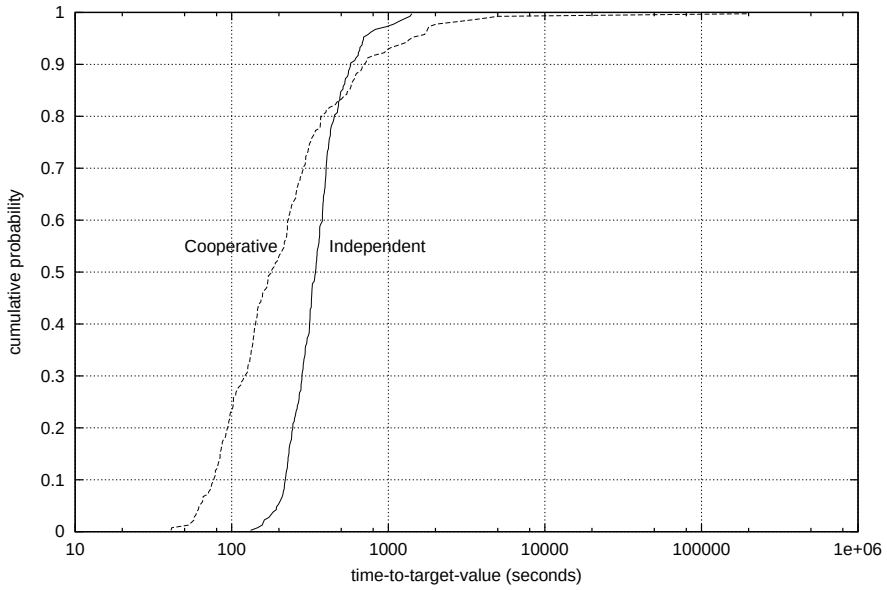


Figure 2.10 Probability distributions of time-to-target-value on an instance of the 2-path network design problem for cooperative and independent parallel implementations of GRASP with path-relinking on a Linux cluster with eight processors.

R.M. Aiex, S.A. Canuto, S.L. Martins, M. Prais, I. Rosseti, M.C. Souza, E. Uchoa, D.S. Vianna, and R.F. Werneck.

Bibliography

R.M. Aiex. *Uma investigação experimental da distribuição de probabilidade de tempo de solução em heurísticas GRASP e sua aplicação na análise de implementações paralelas*. PhD thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, 2002.

R.M. Aiex, S. Binato, and M.G.C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29:393–430, 2003.

R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with path relinking for the three-index assignment problem. Technical report, AT&T Labs Research, Florham Park, NJ 07733, 2000. To appear in *INFORMS J. on Computing*.

R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8:343–373, 2002.

D.J. Aloise, D. Aloise, C.T.M. Rocha, J.D. Melo, and C.C. Ribeiro. Scheduling workover rigs for onshore oil production. Technical report, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, 2003.

E. Balas and M.J. Saltzman. An algorithm for the three-index assignment problem. *Oper. Res.*, 39:150–161, 1991.

S. Binato, H. Faria Jr., and M.G.C. Resende. Greedy randomized adaptive path relinking. In J.P. Sousa, editor, *Proceedings of the IV Metaheuristics International Conference*, pages 393–397, 2001.

R.E. Burkard, R. Rudolf, and G.J. Woeginger. Three-dimensional axial assignment problems with decomposible cost coefficients. *Discrete Applied Mathematics*, 65: 123–139, 1996.

S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local search with perturbation for the prize-collecting Steiner tree problems in graphs. *Networks*, 38:50–58, 2001.

G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: An analytical study of exact and approximate algorithms. *Management Science*, 23:789–810, 1977.

Y. Crama and F.C.R. Spieksma. Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *European Journal of Operational Research*, 60:273–279, 1992.

V.-D. Cung, S.L. Martins, C.C. Ribeiro, and C. Roucairol. Strategies for the parallel implementation of metaheuristics. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 263–308. Kluwer Academic Publishers, 2002.

G. Dahl and B. Johannessen. The 2-path network design problem. Technical Report 292, Institutt for informatikk, Universitete i Oslo, Oslo, Norway, November 2000.

T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.

P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the max-cut problem. *Optimization Methods and Software*, 7:1033–1058, 2002.

P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys on metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

C. Fleurent and F. Glover. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing*, 11: 198–204, 1999.

B. Fortz and M. Thorup. Internet traffic engineering by optimizing ospf weights. In *Proc. IEEE INFOCOM 2000 – The Conference on Computer Communications*, pages 519–528, 2000.

A.M. Frieze. Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research*, 13:161–164, 1983.

M.R. Garey and D.S. Johnson. *Computers and intractability - A guide to the theory of NP-completeness*. W.H. Freeman and Company, 1979.

F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

F. Glover. Multi-start and strategic oscillation methods – Principles to exploit adaptive memory. In M. Laguna and J.L. Gonzáles-Velarde, editors, *Computing Tools*

for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research, pages 1–24. Kluwer, 2000.

F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.

F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

O. Kariv and L. Hakimi. An algorithmic approach to nework location problems, part ii: The *p*-medians. *SIAM Journal of Applied Mathematics*, 37(3):539–560, 1979.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994.

S.L. Martins, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Greedy randomized adaptive search procedures for the steiner problem in graphs. In P.M. Pardalos, S. Rajasekaran, and J. Rolim, editors, *Randomization methods in algorithmic design*, volume 43 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 133–145. American Mathematical Society, 1999.

S.L. Martins, C.C. Ribeiro, and I. Rosseti. Applications and parallel implementations of metaheuristics in network design and routing. In S. Manandhar, editor, *Proceedings of the Asian Applied Computing Conference*, to appear in Lecture Notes in Computer Science. Springer-Verlag, 2004.

C.A. Oliveira, P.M. Pardalos, and M.G.C. Resende. GRASP with path-relinking for the QAP. In Toshihide Ibaraki and Yasunari Yoshitomi, editors, *Proceedings of the Fifth Metaheuristics International Conference*, pages 57–1 – 57–6, 2003.

W.P. Pierskalla. The tri-subsitution method for the three-multidimensional assignment problem. *CORS J.*, 5:71–81, 1967.

M. Prais and C.C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12: 164–176, 2000.

M. G. C. Resende and R. F. Werneck. On the implementation of a swap-based local search procedure for the *p*-median problem. In R. E. Ladner, editor, *Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments (ALENEX'03)*, pages 119–127. SIAM, 2003a.

M.G.C. Resende. Computing approximate solutions of the maximum covering problem using GRASP. *Journal of Heuristics*, 4:161–171, 1998.

M.G.C. Resende, T.A. Feo, and S.H. Smith. Algorithm 787: Fortran subroutines for approximate solution of maximum independent set problems using GRASP. *ACM Transactions on Mathematical Software*, 24:386–394, 1998.

M.G.C. Resende, P.M. Pardalos, and Y. Li. Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software*, 22:104–118, 1996.

M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Fortran subroutines for computing approximate solutions of MAX-SAT problems using GRASP. *Discrete Applied Mathematics*, 100:95–113, 2000.

M.G.C. Resende and C.C. Ribeiro. A GRASP for graph planarization. *Networks*, 29: 173–189, 1997.

M.G.C. Resende and C.C. Ribeiro. A GRASP with path-relinking for private virtual circuit routing. *Networks*, 41:104–114, 2003a.

M.G.C. Resende and C.C. Ribeiro. GRASP and path-relinking: Recent advances and applications. In Toshihide Ibaraki and Yasunari Yoshitomi, editors, *Proceedings of the Fifth Metaheuristics International Conference*, pages T6–1 – T6–6, 2003b.

M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2003c.

M.G.C. Resende and R.F. Werneck. A GRASP with path-relinking for the p-median problem. Technical Report TD-5E53XL, AT&T Labs Research, Florham Park, NJ 07932 USA, 2002a.

M.G.C. Resende and R.F. Werneck. A hybrid heuristic for the p-median problem. Technical Report TD-5NWRCR, AT&T Labs Research, Florham Park, NJ 07932 USA, 2002b.

M.G.C. Resende and R.F. Werneck. A hybrid multistart heuristic for the uncapacitated facility location problem. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ, 2003b.

C.C. Ribeiro and M.G.C. Resende. Algorithm 797: Fortran subroutines for approximate solution of graph planarization problems using GRASP. *ACM Transactions on Mathematical Software*, 25:341–352, 1999.

C.C. Ribeiro and I. Rosseti. A parallel GRASP for the 2-path network design problem. *Lecture Notes in Computer Science*, 2004:922–926, 2002.

C.C. Ribeiro and I. Rosseti. A GRASP with path-relinking heuristic for the 2-path network design problem. Technical report, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, 2004, in preparation.

C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14:228–246, 2002.

C.C. Ribeiro and D.S. Vianna. A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path-relinking. *Revista Tecnologia da Informação*, 3(2):67–70, 2003.

I. Rosseti. *Heurísticas para o problema de síntese de redes a 2-caminhos*. PhD thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, July 2003.

M.C. Souza, C. Duhamel, and C.C. Ribeiro. A GRASP with path-relinking heuristic for the capacitated minimum spanning tree problem. In M.G.C. Resende and J. Souza, editors, *Metaheuristics: Computer Decision Making*, pages 627–658. Kluwer Academic Publishers, 2003.

M. B. Teitz and P. Bart. Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, 16(5):955–961, 1968.

R. Whitaker. A fast algorithm for the greedy interchange of large-scale clustering and median location prolems. *INFOR*, 21:95–108, 1983.

3 LOCAL SEARCH WITH PERTURBATIONS FOR THE PRIZE COLLECTING STEINER TREE PROBLEM IN GRAPHS

Suzana A. Canuto[1], Mauricio G. C. Resende[2] and Celso C. Ribeiro[1]

[1]Department of Computer Science
Catholic University of Rio de Janeiro
Rio de Janeiro, RJ 22453-900 Brazil
canuto@inf.puc-rio.br
celso@inf.puc-rio.br

[2]Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA
mgcr@research.att.com

Abstract: Given an undirected graph with prizes associated with its nodes and weights associated with its edges, the prize-collecting Steiner tree problem consists of finding a subtree of this graph which minimizes the sum of the weights of its edges plus the prizes of the nodes not spanned. In this paper, we describe a multi-start local search algorithm for the prize-collecting Steiner tree problem, based on the generation of initial solutions by a primal-dual algorithm using perturbed node prizes. Path-relinking is used to improve the solutions found by local search and variable neighborhood search is used as a post-optimization procedure. Computational experiments involving different algorithm variants are reported. Our results show that the local search with perturbations approach found optimal solutions on nearly all of the instances tested.
Keywords: Local search, prize collecting, Steiner problem, graphs, path-relinking, variable neighborhood search, network design.

3.1 INTRODUCTION

Let $G = (V,E)$ be a connected undirected graph, where V is the set of nodes and E denotes the set of edges, together with a nonnegative weight function $w : E \rightarrow \mathbb{R}_+$ associated with its edges and a nonnegative prize function $\pi : V \rightarrow \mathbb{R}_+$ associated with its nodes. In the *prize-collecting Steiner tree problem* (PCSTP), one wants to find a subtree of $G = (V,E)$ which minimizes the sum of the weights of its edges plus the prizes of the nodes not spanned.

The prize-collecting Steiner tree problem has an important application in telecommunication local access network design, so as to balance the potential revenue that can be obtained by providing service to customers and the cost to build the network. In this application, a fiber-optic network that provides local service to customers is to be built. In the associated graph, edges are street segments where fiber can be laid. Edge weights are the costs associated with laying the fiber cable along street segments. Nodes in the graph are street intersections and the locations of customer premises, which one can assume are on or near the streets. Node prizes are estimates of the potential revenue to be obtained by providing service to the customers on that node.

We notice that if the subset of nodes X to be spanned is known, we have the *Steiner tree problem*, which consists of finding a minimum weighted connected subtree of G spanning all nodes in X. The Steiner problem in graphs is a classical combinatorial optimization problem, whose decision version has been shown to be NP-complete by Karp (1972). Since the Steiner problem in graphs is a particular case of the PCSTP in which all terminal nodes have infinite prizes and the others have null prizes, then the decision version of PCSTP is also NP-complete.

Construction heuristics and lower bounds based on Lagrangian relaxation for the PCSTP have been proposed by Segev (1987) and Engevall et al. (1998). Both papers report limited computational experience for small graphs having 5 to 100 nodes. More recently, Johnson et al. (2000) described an implementation of the primal-dual 2-approximation algorithm of Goemans and Williamson (1996), for which extensive computational results on large instances are reported.

In this paper, we propose a local search approach with perturbations for finding approximate solutions to the prize-collecting Steiner tree problem and report computational results for large graphs with up to 1000 nodes and 25,000 edges. The approach is based on a neighborhood defined by the set of nodes spanned by the current solution. In the next section, we define the neighborhood structure and the basic iterative improvement algorithm used for local search. The multi-start perturbation algorithm is described in Section 3.3. In Section 3.4 we describe a path-relinking scheme for search intensification and solution improvement. The full algorithm, combining the previous local search with perturbations algorithm and path-relinking with a variable neighborhood search procedure is described in Section 3.5. Computational results are reported in Section 3.6. Concluding remarks are drawn in the last section.

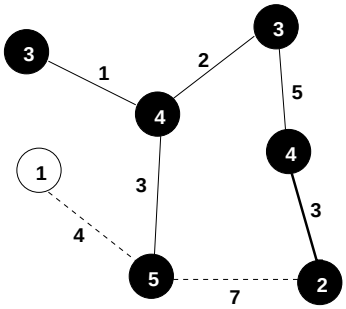


Figure 3.1 Solution $T(X)$ before peeling.

3.2 LOCAL SEARCH AND NEIGHBORHOOD STRUCTURE

Given the graph $G = (V,E)$ together with cost functions $w(\cdot)$ and $\pi(\cdot)$, we associate a solution $T(X)$ of the prize-collecting Steiner tree problem to each subset $X \subseteq V$ of nodes, defined by a minimum spanning tree of the graph induced in G by X. The characteristic vector associated with the solution defined by the subset X of nodes is $v_X \in \{0,1\}^{|V|}$, such that $v_X(i) = 1$ if $i \in X$; $v_X(i) = 0$ otherwise. We define $T_p(X)$ to be the tree obtained from $T(X)$ by recursively eliminating from $T(X)$ all degree-1 nodes whose incident arc has weight greater than its prize. The cost $c(X)$ of solution $T(X)$ is given by the sum of the weights of all edges in $T_p(X)$ plus the sum of the prizes of all nodes not spanned by $T_p(X)$.

We illustrate in Figures 3.1–**??** the peeling procedure leading from tree $T(X)$ to $T_p(X)$. In each figure, nodes in black are those spanned by the current solution using the edges drawn in black. Dashed edges and empty (white) nodes are not part of the solution. Numbers on the edges are edge weights, while the numbers in the nodes are node prizes. Figure 3.1 represents the current solution $T(X)$ whose sum of edge weights and non-spanned node prizes is 15. Since the leaf with node prize two is connected by an incident edge with weight three, it can be removed leading to the tree in Figure 3.2. Once again, the node with prize equal to four in this figure is connected by an edge with larger weight (five) and can then be also removed. The solution $T_p(X)$ obtained by the peeling procedure has cost $c(X) = 13$ and is illustrated in Figure **??**.

The neighborhood $N^{(1)}(X)$ of a solution $T(X)$ is formed by all minimum spanning trees $T(X')$ whose sets X' of nodes differ from X by exactly one node, i.e., $\sum_{i=1}^{i=|V|} |v_{X'}(i) - v_X(i)| = 1$.

A local search approach based on neighborhood structure $N^{(1)}$ starts from some solution $T(X)$ associated with a set X of terminal nodes and progressively replaces it by the first improving move it finds within neighborhood $N^{(1)}(X)$. The algorithmic description of this basic iterative improvement approach used for local search is given in Figure 3.4. The loop from lines 2 to 15 is performed until no further improvement is possible. Lines 3 to 7 control the neighborhood search, which investigates all possible insertions and eliminations of nodes. To speed up the search, the neighborhood investigation starts from the node following that which led to the improving move ob-

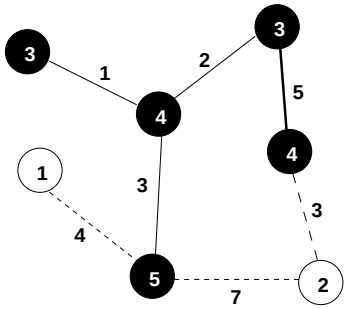


Figure 3.2 Solution after first peeling iteration.

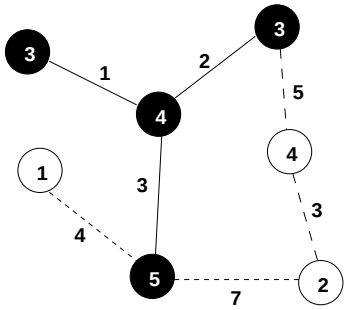


Figure 3.3 Solution $T_p(X)$ after peeling.

tained in the previous iteration (circularity). The cost $c(X')$ of neighbor $T(X')$ has two components and is computed in line 8. The first component is the cost of the minimum spanning tree of the graph induced in G by the nodes of $T_p(X')$, using Kruskal's algorithm (Kruskal, 1956), to which we add the prizes of the nodes not spanned by this tree. Whenever an improving neighbor is found, it replaces the current solution and local search resumes from this new solution (lines 9 to 13). Local search stops when all neighbors have been evaluated and no further reduction in terms of solution cost is possible.

3.3 PERTURBATION ALGORITHM

The perturbation algorithm described in this section is a multi-start approach, with initial solutions provided by the primal-dual algorithm `GW` of Goemans and Williamson (1996). The algorithm is also similar to a GRASP procedure (Feo and Resende, 1995), in which the greedy randomized construction is replaced by the construction of initial solutions using perturbed cost functions.

The basic structure of the local search with perturbations algorithm is presented in Figure 3.5. At each iteration, a new initial solution is constructed by the primal-dual algorithm `GW`. The original node prizes π are used in the first iteration, but they are modified during the forthcoming iterations to enforce diversification of the solutions

```
procedure It_Impr(V,E,w,π,X)
1    local_improvement ← .TRUE.;
2    while local_improvement do
3        local_improvement ← .FALSE.;
4        circfor i = 1,...,|V| while .NOT.local_improvement do
5            if i ∈ X
6            then X' ← X \ {i};
7            else X' ← X ∪ {i};
8            Compute T(X') and its cost c(X');
9            if c(X') < c(X)
10           then do
11               X ← X';
12               local_improvement ← .TRUE.;
13           end_then;
14       end_circfor;
15   end_while;
16   return X;
end It_Impr;
```

Figure 3.4 Pseudo code of the iterative improvement algorithm for local search.

computed by algorithm GW. The spanning tree $T(X)$ associated with the subset X of nodes spanned by the solution constructed in this way is submitted to the local search algorithm It_Impr described in the previous section. The best solution found is updated if it is improved by the current solution. Next, node prizes are updated by a perturbation function, according to the structure of the current solution, and a new initial solution is computed. Two variants of this algorithm are derived, using two different prize perturbation schemes:

• *Perturbation by eliminations*: To enforce search diversification, we drive GW to construct a new solution without some of the nodes appearing in the solution obtained in the previous iteration. This is done by changing to zero the prizes of some persistent nodes which appeared in the solution built by GW and remained at the end of the local search in the previous iteration. In our implementation, we consider a parameter α and we change to zero the prizes associated with a randomly chosen set containing $\alpha\%$ of the persistent nodes observed in the previous iteration.

• *Perturbation by prize changes*: Another strategy to force GW to build different, but still good solutions, consists in introducing some noise into node prizes, similarly to what is proposed by Charon and Hudry (1993), so as to change the objective function. For each node $i \in V$, a perturbation factor β is randomly generated in the interval $[1-a, 1+a]$, where a is an implementation parameter, and the prize associated with node i is changed to $\overline{\pi}(i) = \pi(i) \times \beta$.

```
procedure LS_Perturbations(V, E, w, π)
1    best_value ← +∞;
2    π̄ ← π;
3    for i = 1, ..., max_iterations do
4        X ← GW(V, E, w, π̄);
5        X̄ ← It_Impr(V, E, w, π, X);
6        if c(X̄) < best_value
7        then do
8            X* ← X̄;
9            best_value ← c(X̄);
10       end_then;
11       Compute perturbations and update π̄;
12   end_for;
13   return X*;
end LS_Perturbations;
```

Figure 3.5 Pseudo code of the basic local search with perturbations algorithm.

3.4 ELITE SOLUTIONS AND PATH-RELINKING

The path-relinking approach generates new solutions by exploring trajectories that connect elite solutions (Glover, 1996; Glover and Laguna, 1997). Starting from one or more of these solutions, paths in the solution space leading towards other elite solutions are generated and explored in the search for better solutions. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions. Path-relinking may then be viewed as a strategy that seeks to incorporate attributes of high quality solutions (elite solutions), by favoring these attributes in the selected moves.

Laguna and Martí (1999) adapted the concept of path-relinking for use within a GRASP. This approach can be naturally applied in our local search with perturbations algorithm. A pool of up to *max_pool* high-quality diverse elite solutions is stored to serve as guiding solutions for path-relinking.

Each iteration of algorithm `LS_Perturbations` produces a locally optimal solution $\bar{X}$. A solution $\bar{Y}$ is chosen at random from the elite set and a path of solutions linking $\bar{X}$ to $\bar{Y}$ is constructed by applying a series of changes to the original solution. We first compute the symmetric difference between solutions $\bar{X}$ and $\bar{Y}$, i.e., the set of all nodes appearing in one of them but not in the other. This set defines the moves that should be applied to $\bar{X}$ until the guiding elite solution $\bar{Y}$ is attained. Starting from $\bar{X}$, at each iteration of path-relinking we always perform the best (most decreasing or least increasing) remaining move still in this list until $\bar{Y}$ is attained. The best solution $\tilde{X}$ found along this trajectory replaces the worst solution currently in the pool if it satisfies either one of the following criteria:

- $c(\tilde{X})$ is less than the cost of the best solution currently in the pool;

- $c(\tilde{X})$ is less than the cost of the worst solution currently in the pool and $\tilde{X}$ is sufficiently different from all solutions in the pool. For two solutions to be considered as sufficiently different, the Hamming distance between their characteristic vectors must be greater than a specified nonnegative threshold parameter $\rho \leq 1$ multiplied by the number of vertices in the graph.

3.5 FULL ALGORITHM WITH POST-OPTIMIZATION

In this section, we describe a full algorithm for the PCSTP combining the previous local search with perturbations algorithm and path-relinking with a variable neighborhood search used as a post-optimization step.

The variable neighborhood search (VNS) metaheuristic, proposed by Mladenović and Hansen (1997), is based on the exploration of a dynamic neighborhood model. We use VNS as a post-processing optimization procedure applied to the best solution found by the local search with perturbations algorithm described in Section 3.3.

The k-th order neighborhood $N^{(k)}(X)$ of a solution $T(X)$ is formed by all minimum spanning trees $T(X')$ whose sets X' of nodes differ from X by exactly k nodes, i.e., $\sum_{i=1}^{i=|V|} |v_{X'}(i) - v_X(i)| = k$. Let $k_{\max}$ be the order of the highest neighborhood explored.

```
procedure VNS(V, E, w, π, X)
1    for t = 1, ..., max_trials do
2        k ← 1;
3        while k ≤ k_max do
4            Randomly generate X̂ ∈ N^(k)(X);
5            X̄ ← It_Impr(V, E, w, π, X̂);
6            if c(X̄) < c(X)
7            then do
8                X ← X̄;
9                k ← 1;
10           end_then;
11           else k ← k + 1;
12       end_while
13   end_for
14   return X;
end VNS;
```

Figure 3.6 Pseudo code of the VNS post-optimization procedure.

Figure 3.6 describes the VNS post-optimization procedure. Starting from an initial solution X, the loop from lines 1 to 13 is executed for *max_trials* iterations in which the sequence of neighborhoods $N^{(1)}(X), \ldots, N^{(k_{\max})}(X)$ is explored without finding any improving solution. Each trial starts in line 2 using the first order neighborhood. The loop from lines 3 to 12 explores a sequence of increasing order neighborhoods. A subset $\hat{X}$ of nodes belonging to the k-th order neighborhood leading to a feasible

solution $T(\hat{X})$ is randomly generated in line 4. The `It_Impr` procedure described in Section 3.2 is applied to $\hat{X}$ in line 5. If the locally optimal solution $\bar{X}$ improves upon the current solution X, the latter is updated in line 8 and the algorithm returns to explore again the first order neighborhood in line 9. Otherwise, the search continues using the next order neighborhood (line 11). The best solution found is returned in line 14.

The full algorithm `LS_PCSTP` described in Figure 3.7 incorporates the path-relinking strategy and the VNS post-optimization procedure to algorithm `LS_Perturbations`, presented in Section 3.3. A hashing procedure is also incorporated to avoid the investigation of previously visited solutions. Initializations are performed in lines 1 and 2. The iterations of this multi-start procedure are done in the loop from lines 3 to 19. For each iteration, a new initial solution X is constructed by the primal-dual algorithm `GW` in line 4 using the current node prizes $\bar{\pi}$. In line 5, we use a hashing function to check if the constructed solution is stored in the list of already visited solutions. In case X is a new solution, a locally optimal solution $\bar{X}$ is obtained by the `It_Impr` procedure given in Section 3.2 starting from X. Solution $\bar{X}$ is inserted in the pool of elite solutions in line 8 if either one of the membership conditions described in Section 3.4 ((i) optimality with respect to the solutions currently in the pool, or (ii) quality and diversity) is satisfied. Next, a solution $\bar{Y}$ is randomly selected from the pool of elite solutions in line 9. The path-relinking scheme described in Section 3.4 is applied to $\bar{X}$ using $\bar{Y}$ as the guiding solution, leading to the best solution $\tilde{X}$ in the trajectory (line 10). The incumbent solution and its value are recorded in lines 11 to 15. Once again, in line 16 we check for the insertion of solution $\tilde{X}$ in the pool of elite solutions if either one of the membership conditions is satisfied. In line 18, the node prizes $\bar{\pi}$ are updated for the next iteration, according to a perturbation scheme. Finally, the `VNS` procedure is applied in line 20 to the incumbent solution.

3.6 COMPUTATIONAL RESULTS

Since few benchmark instances for the prize-collecting Steiner tree problem are available, we have generated a set of 80 test problems derived from the Steiner problem instances of the OR-Library (Beasley, 1990). For each of the 40 problems from series C and D, we generated two sets of test instances. All original non terminal nodes receive a null prize. A prize randomly generated in the interval $[1, max\_prize]$ is associated with each original terminal node, where $max\_prize = 10$ for problems in set A and $max\_prize = 100$ for problems in set B. We also used test instances provided in Johnson et al. (2000). Instances in this class are labeled starting with a P or K. In the first group, instances are unstructured and are designed to have constant expected degree and prize to weight ratio. The second group is comprised of random geometric instances designed to have a structure somewhat similar to street maps. A detailed description of the generators for these instances can be found in the above reference.

Algorithm `LS_PCSTP` described in Section 3.5 and all other procedures have been coded in `C` and compiled with `gcc` version 2.7.2.3 under Linux version 2.0.36. Function `elf_hash` from library `libelf` version 0.6.4 (Riepe, 1996) was used for string hashing. All computational experiments were performed on a 400 MHz IBM Pentium II computer with 64 Mbytes of memory.

```
procedure LS_PCSTP(V, E, w, π)
1    best_value ← ∞;
2    π̄ ← π;
3    for i = 1, ..., max_iterations do
4        X ← GW(V, E, w, π̄);
5        if X was not previously visited
6        then do
7            X̄ ← It_Impr(V, E, w, π, X);
8            Insert X̄ in the pool if membership conditions are satisfied;
9            Randomly select a solution Ȳ from the pool of elite solutions;
10           Let X̃ be the best solution found by path-relinking X̄ and Ȳ;
11           if c(X̃) < best_value
12           then do
13               X* ← X̃;
14               best_value ← c(X̃);
15           end_then;
16           Insert X̃ in the pool if membership conditions are satisfied;
17       end_then;
18       Compute perturbations and update π̄;
19   end_for;
20   X* ← VNS(V, E, w, π, X*);
21   return X*;
end LS_PCSTP;
```

Figure 3.7 Pseudo code of the full algorithm for PCSTP.

The perturbation scheme used in algorithm LS_PCSTP (Figure 3.7, line 18) is based on alternating in odd and even iterations the two strategies (perturbation by eliminations and perturbation by prize changes) described in Section 3.3. We have used the following parameter settings in our implementation: $\alpha = 20$, $a = 0.1$, $max\_iterations = 500$, $max\_trials = 10$, $k_{\max} = 35$, $\rho = 0.05$, and $max\_pool = 10$. The random number generator of Schrage (1979) is used. We use an early version of the C implementation of the Goemans and Williamson algorithm developed by Johnson et al. (2000).

We present in Tables 3.1 and 3.2 the computational results for the test problems derived from the instances in the OR-Library. For each instance, this table shows its identification, its number of nodes and edges, the value of the initial solution obtained with the Goemans and Williamson algorithm, the value of the solution obtained with the original prizes after the first iteration of algorithm LS_PCSTP, the value of the solution obtained with algorithm LS_PCSTP without VNS (together with the corresponding computation time in seconds for the 500 iterations), the value of the solution obtained with the full algorithm LS_PCSTP including the VNS post-optimization step (together with the additional computation time in seconds corresponding to the variable neighborhood procedure) for all problems for which the procedure without VNS failed to find a provably optimal solution, a lower bound obtained by solving a linear program-

ming relaxation of an integer programming formulation of the problem (Lucena and Resende, 1999), and an indication of whether this bound is tight. We present the same statistics for the test problems of Johnson et al. (2000) in Table 3.3. On all of these tables, we indicate with bold face whenever a provably optimal solution is found.

Table 3.1 Run statistics on test instances derived from series C OR-Library problems

Problem	Nodes	Edges	GW	It. Impr.	LS+PR	Time	VNS	Time	LB	Opt?
C.01-A	500	625	**18**	**18**	**18**	3			18	y
C.01-B	500	625	89	89	**85**	58			85	y
C.02-A	500	625	**50**	**50**	**50**	7			50	y
C.02-B	500	625	145	145	**141**	54			141	y
C.03-A	500	625	**414**	**414**	**414**	87			414	y
C.03-B	500	625	774	749	**737**	294			737	y
C.04-A	500	625	626	626	**618**	148			618	y
C.04-B	500	625	1106	1068	**1063**	387			1063	y
C.05-A	500	625	1090	1086	**1080**	447			1080	y
C.05-B	500	625	1549	**1528**	**1528**	397			1528	y
C.06-A	500	1000	**18**	**18**	**18**	9			18	y
C.06-B	500	1000	61	57	56	90	**55**	89	55	y
C.07-A	500	1000	**50**	**50**	**50**	34			50	y
C.07-B	500	1000	113	113	103	102	103	65	102	y
C.08-A	500	1000	376	368	**361**	313			361	y
C.08-B	500	1000	543	506	**500**	404			500	y
C.09-A	500	1000	541	537	**533**	475			533	y
C.09-B	500	1000	739	707	**694**	583			694	y
C.10-A	500	1000	870	860	**859**	628			859	y
C.10-B	500	1000	1095	1071	**1069**	474			1069	y
C.11-A	500	2500	**18**	**18**	**18**	128			18	y
C.11-B	500	2500	38	35	**32**	140			32	y
C.12-A	500	2500	40	39	**38**	162			38	y
C.12-B	500	2500	50	47	**46**	156			46	y
C.13-A	500	2500	247	238	237	797	237	253	236	y
C.13-B	500	2500	279	264	**258**	733			258	y
C.14-A	500	2500	309	295	**293**	829			293	y
C.14-B	500	2500	332	319	**318**	766			318	y
C.15-A	500	2500	515	503	**501**	957			501	y
C.15-B	500	2500	572	**551**	**551**	837			551	y
C.16-A	500	12500	15	12	12	429	**11**	1491	11	y
C.16-B	500	12500	15	12	12	292	**11**	1466	11	y
C.17-A	500	12500	20	20	**18**	549			18	y
C.17-B	500	12500	20	20	**18**	434			18	y
C.18-A	500	12500	130	116	**111**	3990			111	y
C.18-B	500	12500	131	118	**113**	3262			113	y
C.19-A	500	12500	165	149	**146**	3928			146	y
C.19-B	500	12500	167	151	**146**	3390			146	y
C.20-A	500	12500	271	266	266	3009	266	1302	263	?
C.20-B	500	12500	274	267	267	2456	267	1344	264	?

We next make some observations concerning the computational results. We refer to the instances derived from the Series C and D test problems of the OR-Library

Table 3.2 Run statistics on test instances derived from series D OR-Library problems

Problem	Nodes	Edges	GW	It. Impr.	LS+PR	Time	VNS	Time	LB	Opt?
D.01-A	1000	1250	**18**	**18**	**18**	6			18	y
D.01-B	1000	1250	108	**106**	**106**	257			106	y
D.02-A	1000	1250	**50**	**50**	**50**	7			50	y
D.02-B	1000	1250	248	241	228	280	228	206	218	y
D.03-A	1000	1250	815	809	**807**	734			807	y
D.03-B	1000	1250	1597	1546	1510	1702	1510	482	1509	y
D.04-A	1000	1250	1211	1205	**1203**	1263			1203	y
D.04-B	1000	1250	1948	1894	1884	1625	**1881**	608	1881	y
D.05-A	1000	1250	2195	2171	2158	2910	**2157**	442	2157	y
D.05-B	1000	1250	3194	3142	3136	1874	**3135**	681	3135	y
D.06-A	1000	2000	**18**	**18**	**18**	20			18	y
D.06-B	1000	2000	75	75	74	401	70	301	67	y
D.07-A	1000	2000	**50**	**50**	**50**	195			50	y
D.07-B	1000	2000	131	118	111	434	105	277	103	y
D.08-A	1000	2000	768	760	**755**	1727			755	y
D.08-B	1000	2000	1121	1060	1038	2571	1038	604	1036	y
D.09-A	1000	2000	1115	1085	1072	3627	1072	482	1070	?
D.09-B	1000	2000	1514	1438	**1420**	2754			1420	y
D.10-A	1000	2000	1735	1684	**1671**	4193			1671	y
D.10-B	1000	2000	2161	2085	**2079**	2644			2079	y
D.11-A	1000	5000	**18**	**18**	**18**	540			18	y
D.11-B	1000	5000	31	31	30	611	30	669	29	y
D.12-A	1000	5000	44	**42**	**42**	844			42	y
D.12-B	1000	5000	44	**42**	**42**	687			42	y
D.13-A	1000	5000	482	453	**445**	5047			445	y
D.13-B	1000	5000	524	495	**486**	4288			486	y
D.14-A	1000	5000	648	607	**602**	6388			602	y
D.14-B	1000	5000	720	681	665	4658	665	1520	664	?
D.15-A	1000	5000	1083	1045	1042	6366	1042	1474	1040	?
D.15-B	1000	5000	1158	1109	1108	3770	1108	1450	1105	?
D.16-A	1000	25000	17	17	**13**	1397			13	y
D.16-B	1000	25000	18	13	**13**	1043			13	y
D.17-A	1000	25000	25	**23**	**23**	3506			23	y
D.17-B	1000	25000	25	**23**	**23**	2089			23	y
D.18-A	1000	25000	261	237	**218**	30044			218	y
D.18-B	1000	25000	269	240	224	29134	224	7509	223	?
D.19-A	1000	25000	353	318	308	34038	308	6917	306	?
D.19-B	1000	25000	362	324	311	31527	311	7073	310	?
D.20-A	1000	25000	550	537	536	21329	536	6810	529	?
D.20-B	1000	25000	554	538	537	15038	537	7066	530	?

as Series C and Series D, respectively. The instances from Johnson et al. (2000) are referred to as Series JMP.

Table 3.4 summarizes for each algorithm variant and for each series, the number of test instances, the number of optimal solutions found, the number of solutions within 1%, 5%, and 10% of the linear programming lower bound, and the largest percentage deviation from this lower bound. We compute percentage deviation of the solution

Table 3.3 Run statistics on test instances from Johnson et al. (2000).

Problem	Nodes	Edges	GW	It. Impr.	LS+PR	Time	VNS	Time	LB	Opt?
P100	100	317	813282	809150	**803300**	15			803300	y
P100.1	100	284	979080	951458	**926238**	14			926238	y
P100.2	100	297	419255	**401641**	**401641**	5			401641	y
P100.3	100	316	685115	**659644**	**659644**	10			659644	y
P100.4	100	284	827789	**827419**	**827419**	10			827419	y
P200	200	587	1347847	1325758	**1317874**	72			1317874	y
P400	400	1200	2567429	2479996	**2459904**	397			2459904	y
P400.1	400	1212	2914063	2826251	**2808440**	382			2808440	y
P400.2	400	1196	2584009	2569964	**2518577**	396			2518577	y
P400.3	400	1175	3060369	2974502	**2951725**	500			2951725	y
P400.4	400	1144	3003004	2881757	2852956	426	2852956	139	2817438	y
K100	100	351	135511	135511	**135511**	2			135511	y
K100.1	100	348	**124108**	**124108**	**124108**	2			124108	y
K100.2	100	339	204904	201816	**200262**	3			200262	y
K100.3	100	407	**115953**	**115953**	**115953**	3			115953	y
K100.4	100	364	88159	88159	88159	2	**87498**	4	87498	y
K100.5	100	358	**119078**	**119078**	**119078**	2			119078	y
K100.6	100	307	**132886**	**132886**	**132886**	2			132886	y
K100.7	100	315	**172457**	**172457**	**172457**	2			172457	y
K100.8	100	343	212236	**210869**	**210869**	2			210869	y
K100.9	100	333	**122917**	**122917**	**122917**	2			122917	y
K100.10	100	319	**133567**	**133567**	**133567**	1			133567	y
K200	200	691	332873	332873	**329211**	9			329211	y
K400	400	1515	352513	**350093**	**350093**	68			350093	y
K400.1	400	1470	494434	492424	491160	87	**490771**	107	490771	y
K400.2	400	1527	483013	479642	478035	125	477892	109	477073	y
K400.3	400	1492	417421	415328	415328	76	415328	64	401881	y
K400.4	400	1426	409243	398250	389705	92	**389451**	112	389451	y
K400.5	400	1456	534287	529581	**519526**	122			519526	y
K400.6	400	1576	375957	**374849**	**374849**	60			374849	y
K400.7	400	1442	491539	486533	478220	194	475130	112	474466	y
K400.8	400	1516	419048	419048	**418614**	42			418614	y
K400.9	400	1500	394026	387142	**383105**	76			383105	y
K400.10	400	1507	410654	408913	396109	125	395848	106	394191	y

value z from the lower bound l_z as $100 \cdot (z - l_z)/l_z$. We first notice that the behavior of the implementation of the Goemans and Williamson 2-approximation algorithm used in our study was much better than its worst case guaranteed performance. With this version of GW, the worst-quality solution found was 38.5% from the optimal value. Recently, Johnson et al. (2000) further developed their implementation, resulting in slight improvements with respect to the version we used in this study. For example, with their new implementation, the deviation from the optimal value of the worst-quality solution was reduced from 38.5% to 18.18%. From the first part of Table 3.4 we observe that the instances from Series JMP are the easiest for algorithm GW, while those of Series D are the hardest. Running times for GW are negligible on these instances. The remainder of Table 3.4 illustrates the effect of adding the following additional components to the

Table 3.4 Performance of GW and successive variants of local search.

Algorithm	Series	Instances	Optimal	$\leq 1\%$	$\leq 5\%$	$\leq 10\%$	Worst
GW	JMP	34	8	15	32	34	6.6%
	C	40	6	7	24	33	36.4%
	D	40	5	7	21	31	38.5%
It. Impr.	JMP	34	14	24	34	34	3.7%
	C	40	8	17	34	40	11.1%
	D	40	11	22	33	37	30.8%
LS+PR	JMP	34	26	32	34	34	3.4%
	C	40	33	35	38	40	9.1%
	D	40	22	34	38	39	10.5%
VNS	JMP	34	29	32	34	34	3.3%
	C	40	36	38	40	40	1.1%
	D	40	25	34	40	40	4.6%

Table 3.5 Improvement of successive variants of local search over GW.

		It. Impr.		LS+PR			VNS		
Series	Instances	# Opt	% impr	# Opt	% impr	Time	# Opt	% impr	Time
JMP	26	6	1.2	18	2.3	157.8	21	2.5	94.1
C	34	2	7.8	27	8.6	956.8	30	10.5	796.6
D	35	6	4.7	17	7.0	7668.1	20	7.5	2749.2

full LS_PCSTP algorithm: iterative improvement applied to the initial solution given by GW with the original node prizes, local search with perturbations and path-relinking, and the full algorithm with the VNS post-optimization step. For all problem series and levels of precision (with respect to the optimal), we notice that each additional component always contributed to increase the number of instances having solutions within that level of precision. This conclusion is further illustrated in Figures 3.8 to 3.10, where the overall results for the three series are depicted. The full algorithm found 29 out of 34, 36 out of 40, and 25 out of 40 optimal solutions for series JMP, C, and D, respectively. The worst-quality solutions found by the full algorithm were 3.3%, 1.1%, and 4.6% above the lower bound for series JMP, C, and D, respectively.

Table 3.5 illustrates, for each series of test problems, the effect of each component of the full LS_PCSTP algorithm. For each series, we first give the number of instances not solved to optimality by GW. For each additional component of the full LS_PCSTP algorithm, the table gives the following statistics for the problems not solved to optimality by GW: number of optimal solutions found, average improvement over the value of the GW solution, and the average computation time in seconds. Computation times for iterative improvement are negligible for these instances.

VNS found three additional optimal solutions for each series. The average improvement of VNS over LS_PCSTP without VNS was about 0.7%. However, the average running times for VNS alone are almost of the same magnitude as those of

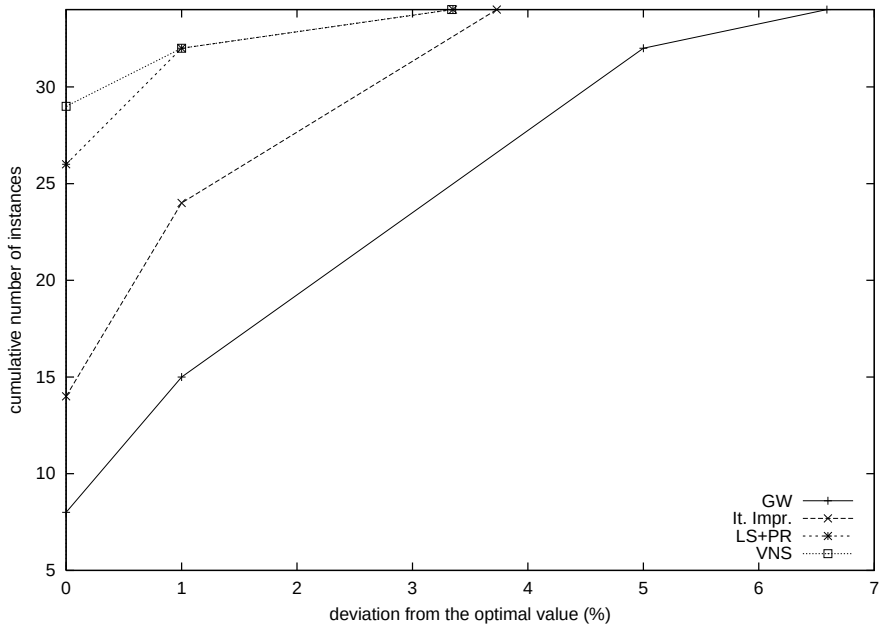


Figure 3.8 Performance of GW and successive variants of local search for Series JMP problems.

LS_PCSTP without VNS. The use of VNS as a post-optimizer would probably be more justified in the case of more difficult or larger problems, for which the other components LS_PCSTP would not be able to find by themselves near-optimal solutions. Note that the average running times in Table 3.5 are inflated by the running times for the six last instances in Tables 3.1 and 3.2 which are derived from notably difficult instances of the Steiner problem in graphs. Compared with the time required to find an optimal solution using the cutting planes algorithm (Lucena and Resende, 1999), the full LS_PCSTP algorithm was in general much faster. For example, instances K400.1 to K400.10 from series JMP required about one day of CPU on a Silicon Graphics Challenge computer, while the full LS_PCSTP algorithm in the worst case took about five minutes on a 400 MHz Pentium II computer for these same instances.

We also note that the full LS_PCSTP algorithm was able to find optimal solutions for all small size problems having 100 nodes within a few seconds of computation time. These problems are comparable with the largest ones considered by Engevall et al. (1998), for which their algorithms did not find optimal solutions in about 9% of the cases. Segev (1987) reported computational experience only for smaller graphs having 5 to 40 nodes.

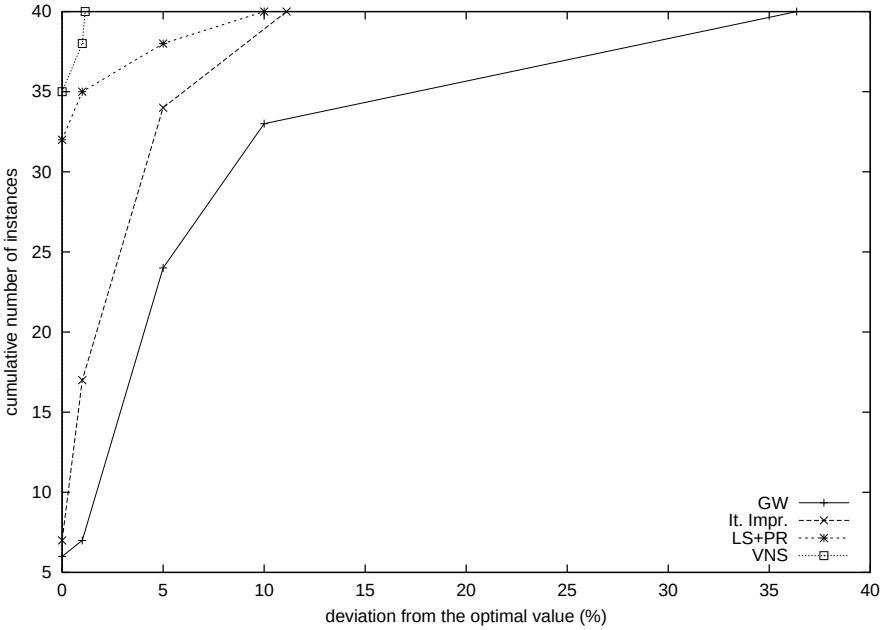


Figure 3.9 Performance of GW and successive variants of local search for Series C problems.

3.7 CONCLUDING REMARKS

In this paper, we have investigated local search strategies for the prize-collecting Steiner tree problem: iterative improvement, multi-start with perturbations, path-relinking, and variable neighborhood search. All strategies have been shown to be effective in improving solutions constructed by the primal-dual 2-approximation algorithm of Goemans and Williamson. Optimal solutions have been found for most test instances. In particular, we stress the effectiveness of path-relinking as an intensification strategy in metaheuristic implementations.

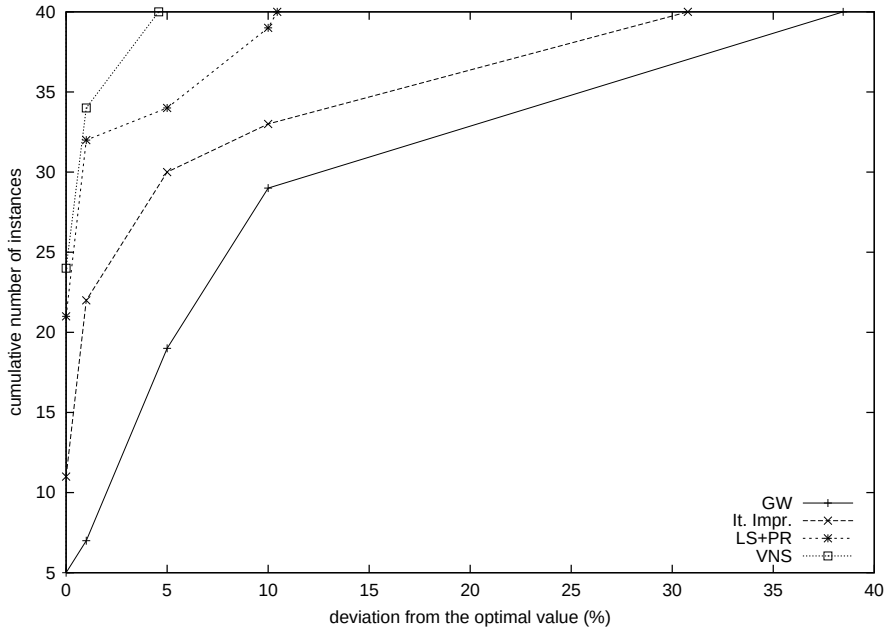


Figure 3.10 Performance of GW and successive variants of local search for Series D problems.

Bibliography

J. E. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

I. Charon and O. Hudry. The noising method: a new method for combinatorial optimization. *Operations Research Letters*, 14:133–137, 1993.

S. Engevall, M. Göthe-Lundgren, and P. Värbrand. A strong lower bound for the node weighted Steiner tree problem. *Networks*, 31:11–17, 1998.

T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

F. Glover. Tabu search and adaptive memory programming – Advances, applications and challenges. Technical report, University of Colorado, 1996.

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

M. X. Goemans and D. P. Williamson. The primal dual method for approximation algorithms and its application to network design problems. In D. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 144–191. PWS Publishing Co., 1996.

D. S. Johnson, M. Minkoff, and S. Phillips. The prize collecting Steiner tree problem: theory and practice. In *Proc. 11th ACM-SIAM Symp. on Discrete Algorithms*, pages 760–769, 2000.

R. M. Karp. Reducibility among combinatorial problems. In E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

J. B. Kruskal. On the shortest spanning tree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, volume 7, pages 48–50, 1956.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS J. on Computing*, 11:44–52, 1999.

A. Lucena and M. G. C. Resende. Strong lower bounds for the prize-collecting Steiner tree problem. Technical report, AT&T Labs Research, 1999.

N. Mladenović and P. Hansen. Variable neighbourhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

M. Riepe. elf_hash, 1996. `http://src.openresources.com/debian/src/libs/HTML/index.html`, online document last visited on February 20, 2001.

L. Schrage. A more portable Fortran random number generator. *ACM Transactions on Mathematical Software*, 5:132–138, 1979.

A. Segev. The node-weighted Steiner tree problem. *Networks*, 17:1–17, 1987.

4 GRASP WITH EVOLUTIONARY PATH-RELINKING

Diogo V. Andrade[1] and Mauricio G. C. Resende[2]

[2]RUTCOR
Rutgers University
Piscataway, NJ 08854 USA
dandrade@rutcor.rutgers.edu

[2]Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA
mgcr@research.att.com

Abstract: We propose GRASP with evolutionary path-relinking, a metaheuristic resulting from the hybridization of GRASP, path-relinking, and evolutionary path-relinking. This metaheuristic is applied to a network migration problem. Experiments show that a GRASP with evolutionary path-relinking heuristic finds solutions faster than a heuristic based on GRASP with path-relinking as well as one based on pure GRASP.
Keywords: GRASP, path-relinking, greedy randomized adaptive path-relinking, evolutionary path-relinking.

4.1 GRASP

GRASP, or greedy randomized adaptive search procedure (Feo and Resende, 1989; 1995), is a metaheuristic for combinatorial optimization. It consists of multiple applications of local search, each starting from a different solution. Starting solutions are generated using some type of greedy randomized construction procedure, such as the semi-greedy algorithm of Hart and Shogan (1987), the sample greedy algorithm of Resende and Werneck (2004), or a perturbation scheme like in Canuto et al. (2001).

4.2 PATH-RELINKING

Path-relinking (PR) is a search strategy that explores trajectories connecting two solutions (Glover, 1996). Given two solutions, their common elements are kept constant and the space of solutions spanned by these elements is searched with the objective of finding a better solution. The size of the solution space grows exponentially with the difference between the two solutions and PR explores only a small portion of the space by moving between the two solutions in a greedy way. One way of carrying out PR is to make one of the solutions the *initial* solution and the other the *target* and move from the initial solution to the target. Ribeiro and Rosseti (2002) introduced *mixed* PR, where the roles of guiding and target solutions are interchanged at each step of the procedure. Faria Jr. et al. (2005) introduced greedy randomized adaptive PR, where instead of moving between the two solutions in a greedy way, the moves are done in a greedy randomized fashion.

4.3 GRASP WITH PATH-RELINKING

Laguna and Martí (2001) proposed integrating GRASP with path-relinking. A pool of elite solutions found in the search is maintained. Membership in the pool is based on quality and diversity, i.e. not only do pool solutions have to be of good quality but they also must be sufficiently different from one another. At the end of each GRASP iteration, PR is applied between the GRASP iterate and a solution chosen at random from the pool. The solution resulting from PR is tested for membership in the elite set. A survey of GRASP with PR is given in Resende and Ribeiro (2005).

4.4 EVOLUTIONARY PATH-RELINKING

Resende and Werneck (2004), introduced evolutionary path-relinking (EvPR) as a post-processing phase for GRASP with PR. In EvPR, the solutions in the pool are evolved as a series of populations $P_1, P_2, \ldots$ of equal size. The initial population (P_0) is the pool of elite solutions produced by GRASP with PR. In iteration k of EvPR, path-relinking is applied between a set of pairs of solutions in population P_k and, with the same rules used to test for membership in the pool of elite solutions, each resulting solution is tested for membership in population P_{k+1}. This evolutionary process is repeated until no improvement is seen from one population to the next.

4.5 GRASP WITH EVOLUTIONARY PATH-RELINKING

Figure 4.1 shows pseudo-code for GRASP with evolutionary path-relinking. GRASP with EvPR is GRASP with PR, where at fixed intervals an intensification of the elite pool is done by way of EvPR. The pool resulting from the EvPR phase is used as the elite pool when GRASP with PR resumes.

We highlight a few important implementation issues for the design of effective EvPR procedures. Since path-relinking between two lower-quality elite solutions is often unfruitful, we limit each iteration of EvPR to path-relinking between the α best elite solutions and the β best elite solutions, where $\alpha \leq \beta$. Since EvPR can be repeatedly applied on the same pool (in consecutive GRASP with PR iterations), we

```
procedure GRASP(i_max, EvPR_freq)
1     Initialize pool P ← ∅
2     for i = 1,...,i_max do
3           x ← GreedyRandomizedConstruction()
4           x ← LocalSearch(x)
5           x ← PathRelinking(x, P)
6           Test x for membership in pool P;
7           if mod(i, EvPR_freq) = 0 then
8                 P ← EvPathRelinking(P);
9           end if
10    end for
11    return(x* ← argmin(P));
end GRASP;
```

Figure 4.1 Pseudo-code for GRASP with EvPR.

make use of greedy randomized adaptive mixed PR with the objective of exploring different trajectories during different applications of EvPR. Lower-quality solutions that have been in the pool for a large number of GRASP iterations should be forced out of the elite pool. To implement this we associate with each elite solution an age. When the solution enters the elite set, its age is zero. At each call of the procedure `EvPathRelinking` in line 8 of the pseudo-code in Figure 4.1 the ages of each pool member are incremented by one. If an elite solution is among the γ worst elite solutions and its age is above a given threshold, then it is removed from the elite set.

4.6 EXPERIMENTAL RESULTS

We illustrate GRASP with EvPR on a network migration scheduling problem, where inter-nodal traffic from an outdated telecommunications network is to be migrated to a new network. Nodes in the old network are deloaded in sequence. All traffic originating, terminating, or passing through the node in the old network is moved to a specific node in the new network. The amount of new capacity needed to achieve the migration depends on the sequence that nodes in the old network are deloaded.

Using real and artificially generated instances, we compare pure GRASP, GRASP with PR, and several variants of GRASP with EvPR. Time-to-target (TTT) plots (Aiex et al., 2006) show some numbers to support the conclusions that GRASP with PR outperforms pure GRASP and that GRASP with EvPR outperforms GRASP with PR.

Bibliography

R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. TTTPLOTS: A perl program to create time-to-target plots. *Optimization Letters*, 2006. published online, DOI 10.1007/s11590-006-0031-4.

S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local search with perturbation for the prize-collecting Steiner tree problems in graphs. *Networks*, 38:50–58, 2001.

H. Faria Jr., S. Binato, M.G.C. Resende, and D.J. Falcão. Transmission network design by a greedy randomized adaptive path relinking approach. *IEEE Transactions on Power Systems*, 20(1):43–49, 2005.

T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.

M. Laguna and R. Martí. A GRASP for coloring sparse graphs. *Computational Optimization and Applications*, 19:165–178, 2001.

M.G.C. Resende and C.C. Ribeiro. GRASP with path-relinking: Recent advances and applications. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*, pages 29–63. Springer, 2005.

M.G.C. Resende and R.F. Werneck. A hybrid heuristic for the p-median problem. *J. of Heuristics*, 10:59–88, 2004.

C.C. Ribeiro and I. Rosseti. A parallel GRASP for the 2-path network design problem. *Lecture Notes in Computer Science*, 2004:922–926, 2002.

II Parallel GRASP

5 TTTPLOTS: A PERL PROGRAM TO CREATE TIME-TO-TARGET PLOTS

Renata M. Aiex[1], Mauricio G. C. Resende[2], and Celso C. Ribeiro[3]

[1]Department of Computer Science
Catholic University of Rio de Janeiro
Rio de Janeiro, RJ 22453-900 Brazil
rma@inf.puc-rio.br

[2]Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA.
mgcr@research.att.com

[3]Department of Computer Science
Universidade Federal Fluminense
Niterói, RJ 24210-240, Brazil
celso@ic.uff.br

Abstract: This chapter describes a perl language program to create time-to-target solution value plots for measured CPU times that are assumed to fit a shifted exponential distribution. This is often the case in local search based heuristics for combinatorial optimization, such as simulated annealing, genetic algorithms, iterated local search, tabu search, WalkSAT, and GRASP. Such plots are very useful in the comparison of different algorithms or strategies for solving a given problem and have been widely used as a tool for algorithm design and comparison. We first discuss how TTT plots are generated. This is followed by a description of the perl program `tttplots.pl`.
Keywords: Time-to-target plots, algorithm engineering, Perl.

5.1 INTRODUCTION

It has been observed that in many implementations of local search based heuristics for combinatorial optimization problems, such as simulated annealing, genetic algorithms, iterated local search, tabu search, WalkSAT, and GRASP (Aiex et al., 2002;

Battiti and Tecchiolli, 1992; Dodd, 1990; Eikelder et al., 1996; Hoos and Stützle, 1999; Hoos, 1999; Osborne and Gillett, 1991; Selman et al., 1994; Taillard, 1991; Verhoeven and Aarts, 1995), the random variable *time to target solution value* is exponentially distributed or fits a two-parameter shifted exponential distribution, i.e. the probability of not having found a given target solution value in t time units is given by $P(t) = e^{-(t-\mu)/\lambda}$, with $\lambda \in \mathbb{R}^+$ and $\mu \in \mathbb{R}$. Hoos and Stützle (1998b) and Hoos and Stützle (1999) conjecture that this is true for all local search based methods for combinatorial optimization.

Time-to-target (TTT) plots display on the ordinate axis the probability that an algorithm will find a solution at least as good as a given target value within a given running time, shown on the abscissa axis. TTT plots were used by Feo et al. (1994) and have been advocated by Hoos and Stützle (1998); Hoos and Stützle (1998a) as a way to characterize the running times of stochastic algorithms for combinatorial optimization.

This paper describes a perl program to create time-to-target plots for measured CPU times that are assumed to fit a shifted exponential distribution. Such plots are very useful in the comparison of different algorithms or strategies for solving a given problem and have been widely used as a tool for algorithm design and comparison. In the next section, we discuss how TTT plots are generated, following closely Aiex et al. (2002). The perl program `tttplots.pl` is described in Section 5.3. The source code is available from the *Electronic Supplementary Material* page of *Optimization Letters*. Section 5.4 presents an example and concluding remarks are made in Section 5.5.

5.2 TIME-TO-TARGET PLOTS

The hypothesis here is that CPU times fit a two parameter, or shifted, exponential distribution. For a given problem instance, we measure the CPU time to find a solution with an objective function value at least as good as a given target value. The heuristic is run n times on the fixed instance and using the given target solution value. For each of the n runs, the random number generator is initialized with a distinct seed and, therefore, the runs are assumed to be independent. To compare the empirical and the theoretical distributions, we follow a standard graphical methodology for data analysis (Chambers et al., 1983). This methodology is used to produce the TTT plots. In the remainder of this section we describe this methodology.

For each instance/target pair, the running times are sorted in increasing order. We associate with the i-th sorted running time t_i a probability $p_i = (i-1/2)/n$, and plot the points $z_i = [t_i, p_i]$, for $i = 1, \ldots, n$. Figure 5.1 illustrates this cumulative probability distribution plot for a instance/target pair obtained by repeatedly applying a GRASP heuristic to find a solution with objective function value at least as good as a given target value. In this figure, we see that the probability of the heuristic finding a solution at least as good as the target value in at most 2 seconds is about 50%, in at most 4 seconds is about 80%, and in at most 6 seconds is about 90%.

The plot in Figure 5.1 appears to fit a shifted exponential distribution. We would like to estimate the parameters of the two-parameter exponential distribution. To do this, we first draw the theoretical quantile-quantile plot (or Q-Q plot) for the data. To describe Q-Q plots, we recall that the cumulative distribution function for the two-

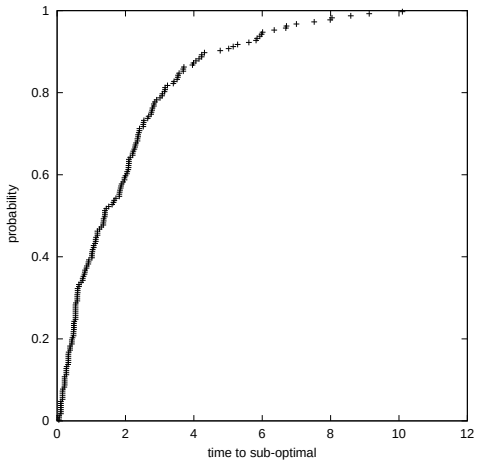


Figure 5.1 Cumulative probability distribution plot of measured data.

parameter exponential distribution is given by $F(t) = 1 - e^{-(t-\mu)/\lambda}$, where λ is the mean of the distribution data (and also is the standard deviation of the data) and μ is the shift of the distribution with respect to the ordinate axis.

For each value p_i, $i = 1,\ldots,n$, we associate a p_i-quantile $Qt(p_i)$ of the theoretical distribution. For each p_i-quantile we have, by definition, that $F((Qt(p_i)) = p_i$. Hence, $Qt(p_i) = F^{-1}(p_i)$ and therefore, for the two-parameter exponential distribution, we have $Qt(p_i) = -\lambda \ln(1-p_i) + \mu$. The quantiles of the data of an empirical distribution are simply the (sorted) raw data.

A theoretical quantile-quantile plot (or theoretical Q-Q plot) is obtained by plotting the quantiles of the data of an empirical distribution against the quantiles of a theoretical distribution. This involves three steps. First, the data (in our case, the measured times) are sorted in ascending order. Second, the quantiles of the theoretical exponential distribution are obtained. Finally, a plot of the data against the theoretical quantiles is made.

In a situation where the theoretical distribution is a close approximation of the empirical distribution, the points in the Q-Q plot will have a nearly straight configuration. In a plot of the data against a two-parameter exponential distribution with $\lambda = 1$ and $\mu = 0$, the points would tend to follow the line $y = \hat{\lambda}x + \hat{\mu}$. Consequently, parameters λ and μ of the two-parameter exponential distribution can be estimated, respectively, by the slope $\hat{\lambda}$ and the intercept $\hat{\mu}$ of the line depicted in the Q-Q plot.

The Q-Q plot shown in Figure 5.2 is obtained by plotting the measured times in the ordinate against the quantiles of a two-parameter exponential distribution with $\lambda = 1$ and $\mu = 0$ in the abscissa, given by $-\ln(1-p_i)$ for $i = 1,\ldots,n$. To avoid possible distortions caused by outliers, we do not estimate the distribution mean with the data mean or by linear regression on the points of the Q-Q plot. Instead, we estimate the slope $\hat{\lambda}$ of the line $y = \lambda x + \mu$ using the upper quartile q_u and lower quartile q_l of the data. The upper and lower quartiles are, respectively, the $Q(1/4)$ and

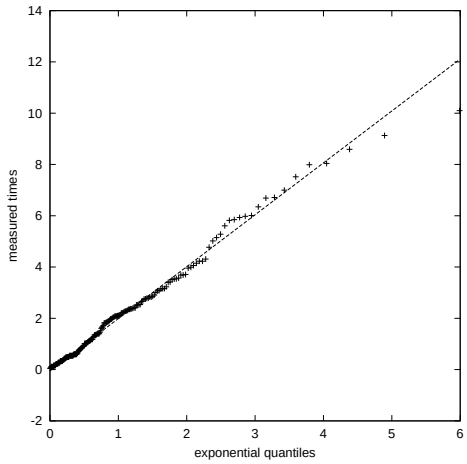


Figure 5.2 Q-Q plot showing fitted line.

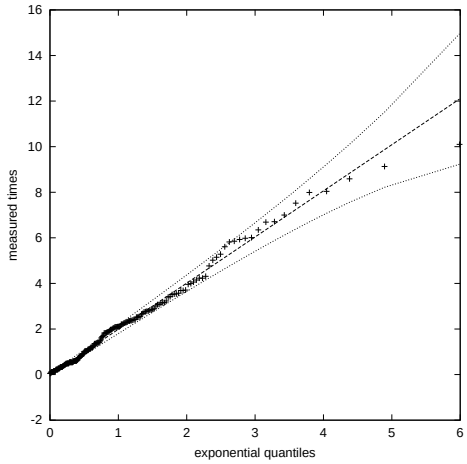


Figure 5.3 Q-Q plot with variability information.

$Q(3/4)$ quantiles. We take $\hat{\lambda} = [z_u - z_l]/[q_u - q_l]$ as an estimate of the slope, where z_u and z_l are the u-th and l-th points of the ordered measured times, respectively. This informal estimation of the distribution of the measured data mean is robust since it will not be distorted by a few outliers (Chambers et al., 1983). Consequently, the estimate for the shift is $\hat{\mu} = z_l - \hat{\lambda} q_l$. To analyze the straightness of the Q-Q plots, we superimpose them with variability information. For each plotted point, we show plus and minus one standard deviation in the vertical direction from the line fitted to the plot. An estimate of the standard deviation for point z_i, $i = 1, \ldots, n$, of the Q-Q plot is

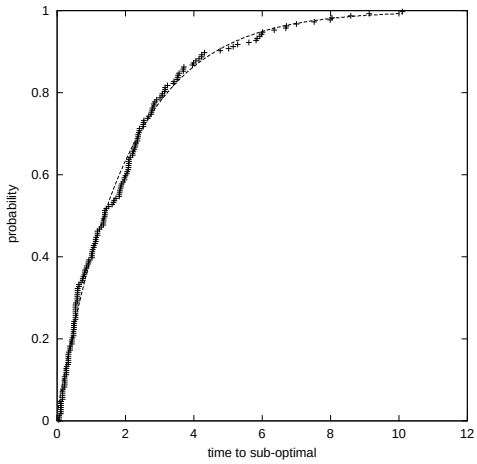


Figure 5.4 Superimposed empirical and theoretical distributions.

$\hat{\sigma} = \hat{\lambda}[p_i/(1-p_i)n]^{\frac{1}{2}}$. Figure 5.3 shows an example of a Q-Q plot with superimposed variability information.

When observing a theoretical quantile-quantile plot with superimposed standard deviation information, one should avoid turning such information into a formal test. One important fact that must be kept in mind is that the natural variability of the data generates departures from the straightness, even if the model of the distribution is valid. The most important reason for portraying standard deviation is that it gives us a sense of the relative variability of the points in the different regions of the plot. However, since one is trying to make simultaneous inferences from many individual inferences, it is difficult to use standard deviations to judge departures from the reference distribution. For example, the probability that a particular point deviates from the reference line by more than two standard deviations is small. However, the probability that at least one of the data points deviates from the line by two standard deviations is probably much greater. In order statistics, this is made more difficult by the high correlation that exists between neighboring points. If one plotted point deviates by more than one standard deviation, there is a good chance that a whole bunch of them will too. Another point to keep in mind is that standard deviations vary substantially in the Q-Q plot, as can be observed in the Q-Q plot in Figure 5.3 that the standard deviation of the points near the high end are substantially larger than the standard deviation of the other end.

Once the two parameters of the distribution are estimated, a superimposed plot of the empirical and theoretical distributions can be made. Figure 5.4 shows this plot corresponding to the Q-Q plot in Figure 5.3.

Table 5.1 Files produced by `tttplots.pl`.

empirical exponential distribution data file	`input_filename-ee.dat`
theoretical exponential distribution data file	`input_filename-te.dat`
empirical QQ-plot data file	`input_filename-el.dat`
theoretical QQ-plot data file	`input_filename-tl.dat`
theoretical upper 1 standard deviation QQ-plot data	`input_filename-ul.dat`
theoretical lower 1 standard deviation QQ-plot data	`input_filename-ll.dat`
theoretical vs empirical TTT plot gnuplot file	`input_filename-exp.gpl`
theoretical vs empirical QQ-plot gnuplot file	`input_filename-qq.gpl`
theoretical vs empirical TTT plot PostScript file	`input_filename-exp.ps`
theoretical vs empirical QQ-plot PostScript file	`input_filename-qq.ps`

5.3 THE PERL PROGRAM

`tttplots.pl` [1] is a perl program that takes as input a file with with CPU times. To be able to produce the plots, `tttplots.pl` requires that `gnuplot` [2] be installed.

To run `tttplots.pl`, simple type: `perl tttplots.pl -f input_filename` where `input_filename.dat` is the input data file with n CPU time data points, one time point per line.

Two plots are produced by `ttplots.pl`:

1. Q-Q plot with superimposed variability information (as in Figure 5.3); and
2. Superimposed empirical and theoretical distributions (as in Figure 5.4).

Besides printing to the standard output some basic statistics of the data file and the estimated parameters, `tttplots.pl` also creates some output files. A list of the files produced by `tttplots.pl` is shown in Table 5.1. Files of type `.dat` contain data points that are plotted by `gnuplot` with files of type `.gpl`. Postscript files of type `.ps` are generated by `gnuplot`.

[1] `tttplots.pl` can be downloaded from `http://www.research.att.com/~mgcr/tttplots`.
[2] `gnuplot` can be downloaded from the gnuplot homepage at `http://www.gnuplot.info`.

5.4 AN EXAMPLE

In this section, we show an example of the plots produced by `tttplots.pl`. We ran the GRASP with path-relinking heuristic for the MAX-CUT problem described in Festa et al. (2002) on instance G13 with a target solution value of 572. We produce plots after 10, 20, 30, 50, 75, 100, 125, 150, and 200 runs. These plots are shown in Figures 5.5, 5.6, and 5.7. These plots were obtained by running `tttplots.pl` using as input files with the CPU times that each of the runs took to find a solution with value at least as good as the target value.

We notice that the larger is the number of runs n (i.e. the number of points plotted), the closer the empirical distribution is to the theoretical distribution. This be seen in the time-to-target plots, as well as in the Q-Q plots. We have observed in practice that using $n = 200$ gives very good approximations of the theoretical distributions. Furthermore, we also notice that the use of "easy" target solution values should be discouraged, since in this case the CPU times are very small in almost all runs and the exponential distribution degenerates to a step function.

5.5 CONCLUDING REMARKS

In this paper, we described a perl language program to create time-to-target plots from a set of running times that are exponentially distributed.

Most time-to-target plots seen in the literature are created from a set of repeated runs of an algorithm on a fixed problem instance. An exception to this was in Feo et al. (1994), where the time-to-target plots were created by running an algorithm a single time on many randomly generated instances having a fixed characteristic (e.g. size and density).

Besides being used to help establish the probability distribution of time-to-target random variables for various stochastic algorithms (Aiex et al., 2003; 2005; Aiex and Resende, 2005; Aiex et al., 2002; Czarnowski and Jędrzejowicz, 2004; Hoos, 1999; Hoos and Stützle, 2000; Resende and Gonzalez Velarde, 2003; Resende and Ribeiro, 2003b; 2005; Stützle and Hoos, 2000), TTT plots have been used in a number of studies to analyze the comparison of

- different heuristics (Aiex et al., 2003; 2005; Buriol et al., 2005; Chiarandini and Stützle, 2002; de Andrade et al., 2005; Fernandes and Ribeiro, 2005; Festa et al., 2005; 2002; Hoos and Boutilier., 2000; Hoos and Stützle, 2000; Marinho, 2005; Oliveira et al., 2004; Resende and Ribeiro, 2003b; 2005; Santos et al., 2004a;b; Silva et al., 2004; Stützle and Hoos, 2000; Tompkins and Hoos, 2003; Tulpan and Hoos, 2003);

- parallel implementations using different number of processors or parallelization strategies (Aiex et al., 2003; 2005; Aiex and Resende, 2005; Martins et al., 2004; Resende and Ribeiro, 2003b; 2005);

- the same algorithm on several instances (Aiex et al., 2003; 2005; Feo et al., 1994; Gent et al., 2003; Hoos and Stützle, 1999; Nudelman et al., 2004; Stützle and Hoos, 2000);

- algorithms using different strategies (Aiex et al., 2005; Martins et al., 2004; Resende and Ribeiro, 2003a;b; 2005; Shmygelska et al., 2002; Shmygelska and Hoos, 2003; Stützle and Hoos, 2000; Tulpan and Hoos, 2003; Tulpan et al., 2003); and
- an algorithm using different parameter settings (Buriol et al., 2005; Hutter, 2004; Hutter et al., 2002; Shmygelska et al., 2002; Shmygelska and Hoos, 2003; Tulpan et al., 2003).

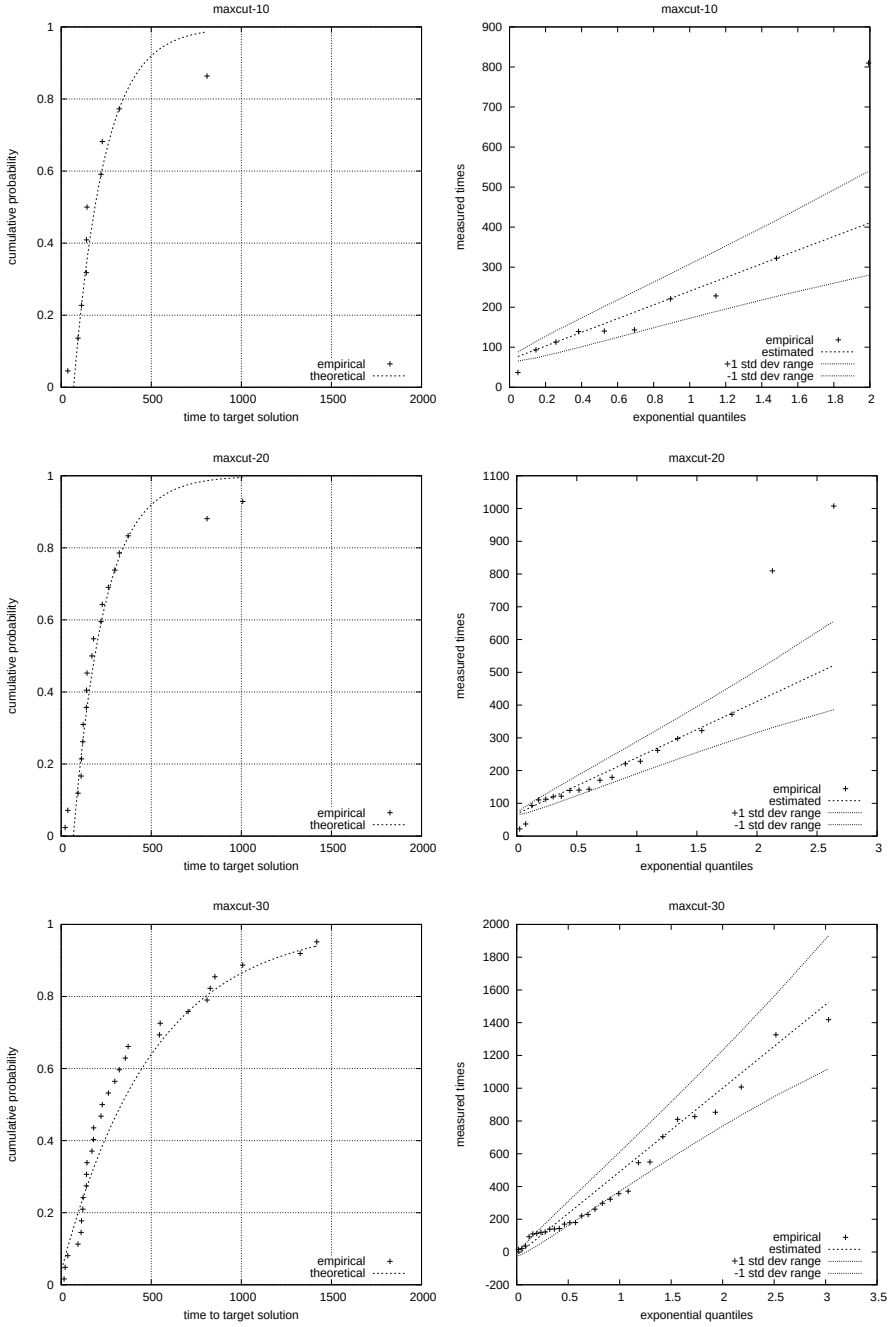


Figure 5.5 Empirical versus theoretical distributions on left and QQ-plots with variability information on right: 10, 20, and 30 data points.

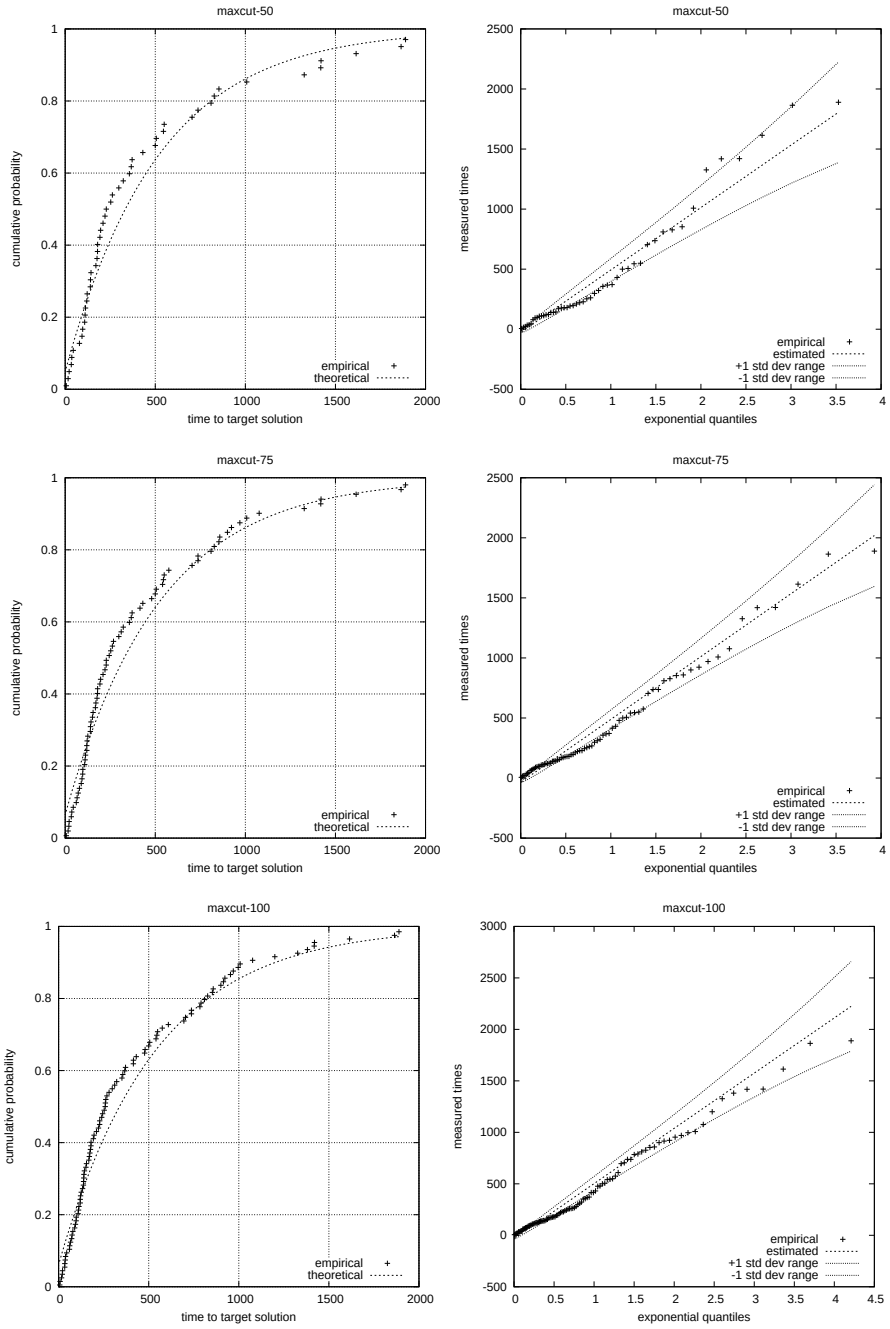


Figure 5.6 Empirical versus theoretical distributions on left and QQ-plots with variability information on right: 50, 75, and 100 data points.

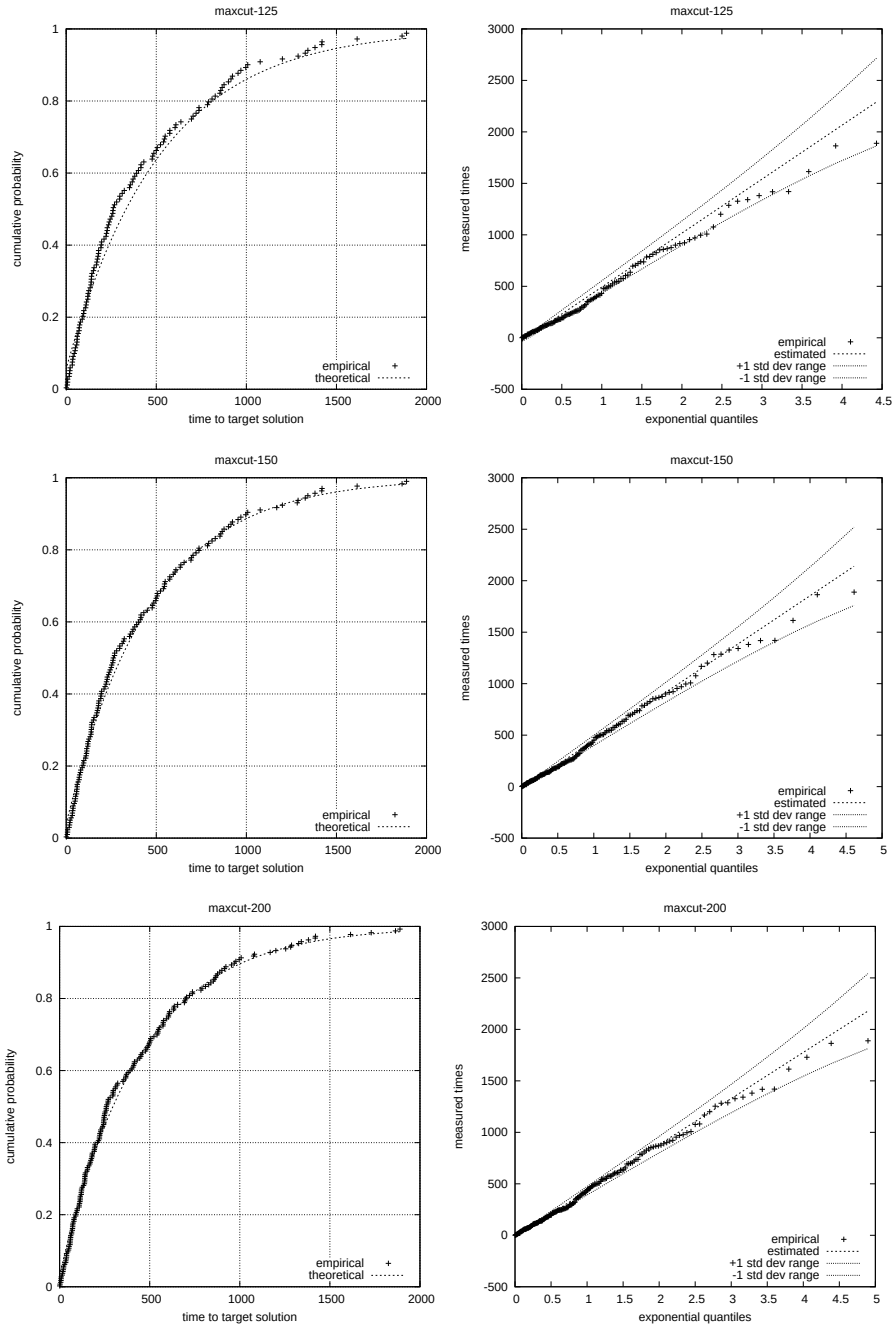


Figure 5.7 Empirical versus theoretical distributions on left and QQ-plots with variability information on right: 125, 150, and 200 data points.

Bibliography

R.M. Aiex, S. Binato, and M.G.C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29:393–430, 2003.

R.M. Aiex, P.M. Pardalos, M.G.C. Resende, and G. Toraldo. GRASP with path relinking for three-index assignment. *INFORMS J. on Computing*, 17:224–247, 2005.

R.M. Aiex and M.G.C. Resende. Parallel strategies for GRASP with path-relinking. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*, pages 301–331. Springer, 2005.

R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *J. of Heuristics*, 8:343–373, 2002.

R. Battiti and G. Tecchiolli. Parallel biased search for combinatorial optimization: Genetic algorithms and TABU. *Microprocessors and Microsystems*, 16:351–367, 1992.

L.S. Buriol, M.G.C. Resende, C.C. Ribeiro, and M. Thorup. A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. *Networks*, 46:36–56, 2005.

J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. *Graphical Methods for Data Analysis*. Chapman & Hall, 1983.

M. Chiarandini and T. Stützle. Experimental evaluation of course timetabeling algorithms. Technical Report AIDA-02-05, Fachgebiet Intellektik, Fachbereich Informatik Technische Universität Darmstadt, 2002.

I. Czarnowski and P. Jędrzejowicz. Probability distribution of solution time in ANN training using population learning algorithm. In L. Rutkowski et al., editor, *ICAISC 2004*, volume 3070 of *Lecture Notes in Artificial Intelligence*, pages 172–177. Springer-Verlag, 2004.

M.R.Q. de Andrade, P.M.F. de Andrade, S.L. Martins, and A. Plastino. GRASP with path-relinking for the maximum diversity problem. In S.E. Nikoletseas, editor,

WEA 2005, volume 3503 of *Lecture Notes in Computer Science*, pages 558–569. Springer-Verlag, 2005.

N. Dodd. Slow annealing versus multiple fast annealing runs: An empirical investigation. *Parallel Computing*, 16:269–272, 1990.

H.M.M. Ten Eikelder, M.G.A. Verhoeven, T.W.M. Vossen, and E.H.L. Aarts. A probabilistic analysis of local search. In I.H. Osman and J.P. Kelly, editors, *Metaheuristics: Theory & applications*, pages 605–618. Kluwer, 1996.

T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.

E.R. Fernandes and C.C. Ribeiro. Using an adaptive memory strategy to improve a multistart heuristic for sequencing by hybridization. In S.E. Nikoletseas, editor, *WEA 2005*, volume 3503 of *Lecture Notes in Computer Science*, pages 4–15. Springer-Verlag, 2005.

P. Festa, P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. GRASP with path-relinking for the weighted maximum satisfiability problem. In S.E. Nikoletseas, editor, *WEA 2005*, volume 3503 of *Lecture Notes in Computer Science*, pages 367–379. Springer-Verlag, 2005.

P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the MAX-CUT problem. *Optimization Methods and Software*, 7:1033–1058, 2002.

I.P. Gent, H.H. Hoos, A.G.D. Rowley, and K. Smyth. Using stochastic local search to solve quantified Boolean formulae. In F. Rossi, editor, *CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, pages 348–362. Springer-Verlag, 2003.

H. Hoos and T. Stützle. On the empirical evaluation of Las Vegas algorithms - Position paper. Technical report, Computer Science Department, University of British Columbia, 1998.

H.H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proc. AAAI-99*, pages 661–666. MIT Press, 1999.

H.H. Hoos and C. Boutilier. Solving combinatorial auctions using stochastic local search. In *Proceedings of the Seventeenth Conference on Artificial Intelligence (AAAI 2000)*, pages 22–29, Austin, Texas, 2000. MIT Press.

H.H. Hoos and T. Stützle. Evaluation Las Vegas algorithms - Pitfalls and remedies. In *Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence*, pages 238–245, 1998a.

H.H. Hoos and T. Stützle. Some surprising regularities in the behaviour of stochastic local search. In M. Maher and J.-F. Puget, editors, *CP'98*, volume 1520 of *Lecture Notes in Computer Science*, page 470. Springer-Verlag, 1998b.

H.H. Hoos and T. Stützle. Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. *Artificial Intelligence*, 112:213–232, 1999.

H.H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation. *J. of Automated Reasoning*, 24:421–481, 2000.

F. Hutter. Stochastic local search for solving the most probable explanation problem in Bayesian networks. Master's thesis, Computer Science Department, Darmstadt University of Technology, 2004.

F. Hutter, D.A.D. Tompkins, and H.H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In P. Van Hentenryck, editor, *CP 2002*, volume 2470 of *Lecture Notes in Computer Science*, pages 233–248. Springer-Verlag, 2002.

E.H. Marinho. Heurísticas busca tabu para o problema de programação de tripulações de ônibus urbanos. Master's thesis, Universidade Federal Fluminense, Niterói, Brazil, 2005. In Portuguese.

S.L. Martins, C.C. Ribeiro, and I. Rosseti. Applications and parallel implementations of metaheuristics in network design and routing. In S. Manandhar et al., editor, *AACC 2004*, volume 3285 of *Lecture Notes in Computer Science*, pages 205–213. Springer-Verlag, 2004.

E. Nudelman, K. Leyton-Brown, H.H. Hoos, A. Devkar, and Y. Shoham. Understanding random SAT: Beyond the clauses-to-variables ratio. In M. Wallace, editor, *CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 438–452. Springer-Verlag, 2004.

C.A.S. Oliveira, P.M. Pardalos, and M.G.C. Resende. GRASP with path-relinking for the quadratic assignment problem. In C.C. Ribeiro and S.L. Martins, editors, *Efficient and Experimental Algorithms – WEA2004*, volume 3059 of *Lecture Notes in Computer Science*, pages 356–368. Springer-Verlag, 2004.

L.J. Osborne and B.E. Gillett. A comparison of two simulated annealing algorithms applied to the directed Steiner problem on networks. *ORSA J. on Computing*, 3: 213–225, 1991.

M.G.C. Resende and J.L. Gonzalez Velarde. GRASP: Procedimientos de búsqueda miope aleatorizado y adaptativo. *Inteligencia Artificial*, 19:61–76, 2003.

M.G.C. Resende and C.C. Ribeiro. A GRASP with path-relinking for private virtual circuit routing. *Networks*, 41:104–114, 2003a.

M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer, 2003b.

M.G.C. Resende and C.C. Ribeiro. Parallel Greedy Randomized Adaptive Search Procedures. In E. Alba, editor, *Parallel Metaheuristics: A new class of algorithms*, pages 315–346. Wiley, 2005.

H.G. Santos, L.S. Ochi, and J.F. Souza M. A tabu search heuristic with efficient diversification strategies for the class/teacher timetabling problem. In E.K. Burke and M. Trick, editors, *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT '04)*, pages 343–358, 2004a.

H.G. Santos, L.S. Ochi, and M.J.F. Souza. An efficient tabu search heuristic for the school timetabling problem. In C.C. Ribeiro and S.L. Martins, editors, *Efficient and Experimental Algorithms – WEA2004*, volume 3059 of *Lecture Notes in Computer Science*, pages 468–481. Springer-Verlag, 2004b.

B. Selman, H.A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the AAAI-94*, pages 337–343. MIT Press, 1994.

A. Shmygelska, R. Aguirre-Hernández, and H.H. Hoos. An ant colony optimization algorithm for the 2D HP protein folding problem. In M. Dorigo et al., editor, *ANTS 2002*, volume 2463 of *Lecture Notes in Computer Science*, pages 40–52. Springer-Verlag, 2002.

A. Shmygelska and H.H. Hoos. An improved ant colony optimisation algorithm for the 2D HP protein folding problem. In Y. Xiang and B. Chaib-draa, editors, *AI 2003*, volume 2671 of *Lecture Notes in Artificial Intelligence*, pages 400–417. Springer-Verlag, 2003.

G.C. Silva, L.S. Ochi, and S.L. Martins. Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem. In C.C. Ribeiro and S.L. Martins, editors, *Efficient and Experimental Algorithms – WEA2004*, volume 3059 of *Lecture Notes in Computer Science*, pages 498–512. Springer-Verlag, 2004.

T. Stützle and H.H. Hoos. Analyzing the run-time behaviour of iterated local search for the TSP. Technical Report IRIDIA/2000-01, IRIDIA, Université Libre de Bruxelles, 2000.

E.D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.

D.A.D. Tompkins and H.H. Hoos. Scaling and probabilistic smoothing: Dynamic local search for unweighted MAX-SAT. In Y. Xiang and B. Chaib-draa, editors, *AI 2003*, volume 2671 of *Lecture Notes in Artificial Intelligence*, pages 145–159. Springer-Verlag, 2003.

D.C. Tulpan and H.H. Hoos. Hybrid randomised neighbourhoods improve stochastic local search for DNA code design. In Y. Xiang and B. Chaib-draa, editors, *AI 2003*, volume 2671 of *Lecture Notes in Artificial Intelligence*, pages 418–433. Springer-Verlag, 2003.

D.C. Tulpan, H.H. Hoos, and E. Condon A. Stochastic local search algorithms for DNA word design. In M. Hagiya and A. Ohuchi, editors, *DNA8*, volume 2568 of *Lecture Notes in Computer Science*, pages 229–241. Springer-Verlag, 2003.

M.G.A. Verhoeven and E.H.L. Aarts. Parallel local search. *J. of Heuristics*, 1:43–66, 1995.

Appendix: Program listing

```
#!/usr/bin/perl
##
#       ----------------------------------------------------------------------------
##
##      tttplots: A Perl program for generating empirical vs
##                theoretical distributions of time-to-target-value
##                plot and corresponding QQ-plots with variability
##                information.
##
##      usage: perl tttplots.pl -f <input-file>
##
##             where <input-file>.dat is the input file of time to
##                   target values (one per line).
##
##
##      authors: Renata M. Aiex, Mauricio G. C. Resende, and
##               Celso C. Ribeiro
##
#       ----------------------------------------------------------------------------
##


##
#       ----------------------------------------------------------------------------
##      Input file name.
#       ----------------------------------------------------------------------------
##
        $datafilethere=0;
        while ($ARGV[0]) {
              if ($ARGV[0] eq "-f") {
                     shift(@ARGV);
                     $filename = $ARGV[0];
                     $datafilename = $filename . ".dat";
                     $datafilethere=1;
                     shift;
              }

        }
        if ($datafilethere == 0) {
              die "Error, data file missing. \nUsage:
                     perl tttplots.pl -f datafile.dat -o outputfile.out \n";
        }

##
#       ----------------------------------------------------------------------------
#       Name output files.
#       ----------------------------------------------------------------------------
##
        $emp_lin_filename = $filename . "-el" . ".dat";
        $the_lin_filename = $filename . "-tl" . ".dat";
        $up_lin_filename = $filename . "-ul" . ".dat";
        $lo_lin_filename = $filename . "-ll" . ".dat";
```

```
        $gpl_lin_filename = $filename . "-qq" . ".gpl";
        $ps_lin_filename = $filename . "-qq" . ".ps";
        $emp_exp_filename = $filename . "-ee" . ".dat";
        $the_exp_filename = $filename . "-te" . ".dat";
        $gpl_exp_filename = $filename . "-exp" . ".gpl";
        $ps_exp_filename = $filename . "-exp" . ".ps";

##
#       ------------------------------------------------------------------------------
#       Open input data file.
#       ------------------------------------------------------------------------------
##
        open (DATFILE,$datafilename) ||
               die "Cannot open file: $datafilename \n";

##
#       ------------------------------------------------------------------------------
#       Read input data file.  Require that data file have one value per line.
#       ------------------------------------------------------------------------------
##
        $n=0;
        while ($line = <DATFILE>){
              chomp($line);
              @field = split(/\s+/,$line);

              $nfields=0;                 #
              foreach $fld (@field){      # count number of fields
                     $nfields++;          #
              }                           #

              if ($nfields != 1){
                     die "Number of fields in data file must be 1 \n";
              }
              $time_value[$n] = $field[0];
              $n++;
        }
        close (DATFILE);

##
#       ------------------------------------------------------------------------------
#       Sort times.
#       ------------------------------------------------------------------------------
##
        @sorted_time_value = sort { $a <=> $b } @time_value;

##
#       ------------------------------------------------------------------------------
#       Write file information to standard output.
#       ------------------------------------------------------------------------------
##
        print "\@-------------------------------------------------------@\n";
        print " tttplots: TIME TO TARGET (TTT) DISTRIBUTION PLOTS \n\n";
        print " Input data set > \n\n";
        print "    data file   : $datafilename \n\n";
        print "    data points : $n \n";
        print "    max value   : $sorted_time_value[$n-1] \n";
##
```

```
#       ------------------------------------------------------------------------------
#       Compute and write input data mean to standard output.
#       ------------------------------------------------------------------------------
##
        $avg=0;
        for ($k=0; $k < $n; $k++){
              $avg=$avg + $sorted_time_value[$k];
        }
        $avg=$avg/$n;
        print "    avg value   : $avg \n";
        print "    min value   : $sorted_time_value[0] \n\n";

##
#       ------------------------------------------------------------------------------
#       Compute probabilities for distribution plot.
#       ------------------------------------------------------------------------------
##
        $nn = 0;
        $np1=$n+1;
        while ($nn < $n){
                $prob[$nn] = $nn + .5;
                $prob[$nn] = $prob[$nn] / $np1;
              $nn++;
        }

##
#       ------------------------------------------------------------------------------
#       Compute distribution parameters.
#       ------------------------------------------------------------------------------
##
        $fq = ($np1 * .25);
        $tq = ($np1 * .75);
        $fq = int($np1 * .25);
        $tq = int($np1 * .75);

        $y = $prob[$fq];
        $zl = $sorted_time_value[$fq];
        $ql = -log(1-$y);
        $y = $prob[$tq];
        $zu = $sorted_time_value[$tq];
        $qu = -log(1-$y);

        $lambda = ($zu - $zl)/($qu - $ql);
        $mu = $zl - ($lambda * $ql);
##
#       ------------------------------------------------------------------------------
#       Write distribution parameters to standard output.
#       ------------------------------------------------------------------------------
##
        print " Estimated parameters (theoretical shifted exponential
                distribution) > \n\n";
        print "    shift (mu)         : $mu \n";
        print "    std. dev. (lambda) : $lambda \n";
        $shifted_mean = $mu+$lambda;
        print "    mean (shifted)     : $shifted_mean  \n";
##
#       ------------------------------------------------------------------------------
```

```
#       Compute theoretical plot (400 points).
#       ------------------------------------------------------------------------------
##
        $tmax = $sorted_time_value[$n-1];
        $inv_lambda = 1/$lambda;
        $eps = $tmax/400;
        $nn = 1;
        while ($nn <= 400){
                $theory_t[$nn-1]= $eps * $nn;
                $theory_p[$nn-1] = 1-exp(-$inv_lambda*($eps * $nn - $mu));
                $nn++;
        }

##
#       ------------------------------------------------------------------------------
#       Compute theoretical time values.
#       ------------------------------------------------------------------------------
##
        $nn = 0;
        while ($nn < $n){
                $theoretical_time[$nn] = -log(1-$prob[$nn]);
                $nn++;
        }

##
#       ------------------------------------------------------------------------------
#       Compute qqplot line, lower and upper error lines.
#       ------------------------------------------------------------------------------
##
        $nn = 0;
        while ($nn < $n){
                $pi = $prob[$nn];
                $x[$nn] = -log(1-$pi);
                $qq_err[$nn] = $lambda * $x[$nn] + $mu;
                $dev = $lambda  * (sqrt($pi/((1-$pi)*$np1)));
                $lo_error_point[$nn] = $qq_err[$nn] - $dev;
                $up_error_point[$nn] = $qq_err[$nn] + $dev;
                $nn++;
        }

##
#       ------------------------------------------------------------------------------
#       Write output files ...
#       ------------------------------------------------------------------------------
##
        open (EMP_LIN_FILE,">$emp_lin_filename") ||
                    die "Cannot open file: $emp_lin_filename \n";
        open (UP_LIN_FILE,">$up_lin_filename") ||
                    die "Cannot open file: $up_lin_filename \n";
        open (LO_LIN_FILE,">$lo_lin_filename") ||
                    die "Cannot open file: $lo_lin_filename \n";
        open (EMP_EXP_FILE,">$emp_exp_filename") ||
                    die "Cannot open file: $emp_exp_filename \n";
        open (THE_LIN_FILE,">$the_lin_filename") ||
                    die "Cannot open file: $the_lin_filename \n";

        $nn = 0;
```

```
        while ($nn < $n){
                print EMP_EXP_FILE
                     "$sorted_time_value[$nn] $prob[$nn] \n";
                print EMP_LIN_FILE
                     "$theoretical_time[$nn] $sorted_time_value[$nn] \n";
                print LO_LIN_FILE "$x[$nn]  $lo_error_point[$nn] \n";
                print UP_LIN_FILE "$x[$nn]  $up_error_point[$nn] \n";
                print THE_LIN_FILE "$x[$nn]  $qq_err[$nn] \n";
                $nn++;
        }
        close (EMP_EXP_FILE);

##
#       ------------------------------------------------------------------------------
#       Theoretical exponential distribution file.
#       ------------------------------------------------------------------------------
##
        open (THE_EXP_FILE,">$the_exp_filename") ||
                die "Cannot open file: $the_exp_filename \n";
        $nn = 0;
        while ($nn < 400){
              print THE_EXP_FILE "$theory_t[$nn] $theory_p[$nn] \n";
              $nn++;
        }

##
#       ------------------------------------------------------------------------------
#       Create qqplot gnuplot file.
#       ------------------------------------------------------------------------------
##
        open (GPL_LIN_FILE,">$gpl_lin_filename") ||
                       die "Cannot open file: $gpl_lin_filename \n";
        print GPL_LIN_FILE <<EOF;
              set xlabel \'exponential quantiles\'
              set size ratio 1
              set ylabel \'measured times\'
              set key right bottom
              set title \'$filename\'
              set terminal postscript color \'Helvetica\'
              set output \'$ps_lin_filename\'
              plot "$emp_lin_filename" t "empirical" w points,
                     "$the_lin_filename" t "estimated" with lines 3,
                     "$up_lin_filename" t "+1 std dev range" w lines 4,
                     "$lo_lin_filename" t "-1 std dev range" w lines 4
              quit
EOF
##
#       ------------------------------------------------------------------------------
#       Create qqplot postscript graphic file.
#       ------------------------------------------------------------------------------
##
        open (PS_EXP_FILE,">$ps_exp_filename") ||
                     die "Cannot open file: $ps_exp_filename \n";
        system("gnuplot $gpl_lin_filename") == 0 ||
                     die "gnuplot (needed for plotting) not found \n";
##
#       ------------------------------------------------------------------------------
```

```
#       Create empirical-theoretical distributions gnuplot file.
#       ------------------------------------------------------------------------------
##
        open (GPL_EXP_FILE,">$gpl_exp_filename") ||
                die "Cannot open file: $gpl_exp_filename \n";
        print GPL_EXP_FILE <<EOF;
              set xlabel \'time to target solution\'
              set size ratio 1
              set ylabel \'cumulative probability\'
              set yrange [0:1]
              set key right bottom
              set grid
              set title \'$filename\'
              set terminal postscript color \'Helvetica\'
              set output \'$ps_exp_filename\'
              plot "$emp_exp_filename" t "empirical" w points,
                     "$the_exp_filename" t "theoretical" w lines 3
              quit
EOF
##
#       ------------------------------------------------------------------------------
#       Create empirical-theoretical distributions postscript
#       graphic file.
#       ------------------------------------------------------------------------------
##
        open (PS_EXP_FILE,">$ps_exp_filename") ||
                 die "Cannot open file: $ps_exp_filename \n";
        system("gnuplot $gpl_exp_filename") == 0 ||
                     die "gnuplot (needed for plotting) not found \n";
##
#       ------------------------------------------------------------------------------
#       Write file names to standard output.
#       ------------------------------------------------------------------------------
##
        print "\n Output data files > \n\n";
        print "    empirical exponential distribution data  : $emp_exp_filename  \n";
        print "    theoretical exponential distribution data: $the_exp_filename  \n";
        print "    empirical qq-plot data                   : $emp_lin_filename\n";
        print "    theoretical qq-plot data                 : $the_lin_filename\n";
        print "    theoretical upper 1 std dev qq-plot data : $up_lin_filename\n";
        print "    theoretical lower 1 std dev qq-plot data : $lo_lin_filename\n";
        print "    theor. vs empir. ttt plot gnuplot file   : $gpl_exp_filename\n";
        print "    theor. vs empir. qq-plot gnuplot file    : $gpl_lin_filename\n";
        print "    theor. vs empir. ttt plot postscript file: $ps_exp_filename\n";
        print "    theor. vs empir. qq-plot postscript file : $ps_lin_filename\n";
        print "\n DONE \n";
        print "\@-------------------------------------------------------@\n";
        print "\n";
##
#       ------------------------------------------------------------------------------
#       End of program.
#       ------------------------------------------------------------------------------
##
```

6 PROBABILITY DISTRIBUTION OF SOLUTION TIME IN GRASP: AN EXPERIMENTAL INVESTIGATION

Renata M. Aiex[1], Mauricio G. C. Resende[2], and Celso C. Ribeiro[1]

[1]Department of Computer Science
Catholic University of Rio de Janeiro
Rio de Janeiro, RJ 22453-900 Brazil
rma@inf.puc-rio.br
celso@ic.uff.br

[2]Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA.
mgcr@research.att.com

Abstract: A GRASP (greedy randomized adaptive search procedure) is a multi-start metaheuristic for combinatorial optimization. We study the probability distributions of solution time to a sub-optimal target value in five GRASPs that have appeared in the literature and for which source code is available. The distributions are estimated by running 12,000 independent runs of the heuristic. Standard methodology for graphical analysis is used to compare the empirical and theoretical distributions and estimate the parameters of the distributions. We conclude that the solution time to a sub-optimal target value fits a two-parameter exponential distribution. Hence, it is possible to approximately achieve linear speed-up by implementing GRASP in parallel.
Keywords: GRASP, probability distribution of running time, time-to-target plots.

6.1 INTRODUCTION

A greedy randomized adaptive search procedure (GRASP) (Feo and Resende, 1989; 1995; Festa and Resende, 2000) is a multi-start or iterative process, in which each GRASP iteration consists of two phases. In a construction phase, a feasible solution

Table 6.1 CPU time (in seconds) and speed-up on MAX-SAT problems. Average speed-up is shown for 5, 10, and 15 processors.

	1 processor	5 processors		10 processors		15 processors	
problem	time	time	speed-up	time	speed-up	time	speed-up
jnh201	310.4	62.8	4.9	30.5	10.2	22.2	14.0
jnh202	312.2	59.8	5.2	31.2	10.0	23.4	13.3
jnh203	351.2	72.3	4.9	35.2	10.0	23.2	15.1
jnh205	327.8	63.4	5.2	32.1	10.2	22.5	14.6
jnh207	304.7	56.7	5.4	29.6	10.3	19.8	15.4
jnh208	355.2	65.6	5.4	33.2	10.7	21.0	16.9
jnh209	339.0	60.5	5.6	33.6	10.1	21.6	15.7
jnh210	318.5	57.6	5.5	32.5	9.8	20.8	15.3
jnh301	414.5	85.3	4.9	45.2	9.2	28.3	14.6
jnh302	398.7	88.6	4.5	48.2	8.3	27.0	14.7
average speed-up:			5.2		9.9		15.0

is produced and in a local search phase, a local optimum in the neighborhood of the constructed solution is sought. The best overall solution is kept as the result.

In the construction phase, a feasible solution is iteratively constructed, one element at a time. The basic GRASP construction phase is similar to the semi-greedy heuristic proposed independently by Hart and Shogan (1987). At each construction iteration, the choice of the next element to be added is determined by ordering all candidate elements (i.e. those that can be added to the solution) in a candidate list C with respect to a greedy function $g : C \rightarrow \mathbb{R}$. This function measures the (myopic) benefit of selecting each element. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but not necessarily the top candidate. The list of best candidates is called the *restricted candidate list* (RCL).

It is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution in the neighborhood of the current solution. It terminates when no better solution is found in the neighborhood.

As with any multi-start heuristic for combinatorial optimization, a GRASP can be implemented in parallel by dividing the k independent iterations among ρ processors. Another approach is to give a target value τ of the objective function to each processor and execute the algorithm until the first processor finds a solution with value at least as good as τ, at which point all processors halt. Some care is needed to assure that no two iterations start with identical random number generator seeds (Pardalos et al., 1996). These are examples of *multiple independent walk* parallelism (Verhoeven and Aarts, 1995).

Many parallel implementations of GRASP using the above strategies have been reported in the literature, e.g. (Martins et al., 2000; 1998; Murphey et al., 1998; Pardalos et al., 1995; 1996). In most of these papers, a common observation was made. The speedups in the measured running times were proportional to the number of processors. A typical example can be seen in Pardalos et al. (1996) where, for a PVM-based implementation of a parallel GRASP for the MAX-SAT, average speed-ups almost identical to the number of processors were measured (see Table 6.1).

This observation can be explained if the random variable *solution time to target* is exponentially distributed, as indicated by the following proposition (Verhoeven and Aarts, 1995).

Proposition 1 *Let $P_\rho(t)$ be the probability of not having found a given (target) solution in t time units with ρ independent processes. If $P_1(t) = e^{-t/\lambda}$ with $\lambda \in \mathbb{R}^+$, i.e. P_1 corresponds to an exponential distribution, then $P_\rho(t) = e^{-\rho t/\lambda}$.*

The above proposition follows from the definition of the exponential distribution. It implies that the probability of finding a solution of a given value in time ρt with a sequential process is equal to the probability of finding a solution at least as good as that given value in time t with ρ independent parallel processes. Hence, it is possible to achieve linear speed-up in solution time to target solution by multiple independent processes.

An analogous proposition can be stated for a two parameter (shifted) exponential distribution.

Proposition 2 *Let $P_\rho(t)$ be the probability of not having found a given (target) solution in t time units with ρ independent processes. If $P_1(t) = e^{-(t-\mu)/\lambda}$ with $\lambda \in \mathbb{R}^+$ and $\mu \in \mathbb{R}$, i.e. P_1 corresponds to a two parameter exponential distribution, then $P_\rho(t) = e^{-\rho(t-\mu)/\lambda}$.*

Analogously, this proposition follows from the definition of the two parameter exponential distribution. It implies that the probability of finding a solution of a given value in time ρt with a sequential process is equal to $1 - e^{-(\rho t-\mu)/\lambda}$ while the probability of finding a solution at least as good as that given value in time t with ρ independent parallel processes is $1 - e^{-\rho(t-\mu)/\lambda}$. Note that if $\mu = 0$, then both probabilities are equal and correspond to the non-shifted exponential distribution. Furthermore, if $\rho\mu \ll \lambda$, then the two probabilities are approximately equal and it is possible to approximately achieve linear speed-up in solution time to target solution by multiple independent processes.

This behavior has been noted in a number of metaheuristics. These include simulated annealing (Dodd, 1990; Osborne and Gillett, 1991); iterated local search algorithms for the traveling salesman problem (Eikelder et al., 1996), where it is shown that the probability of finding a sub-optimal solution is given by a shifted exponential distribution, allowing for the time to find the first local optimum; tabu search, provided that the search starts from a local optimum (Battiti and Tecchiolli, 1992; Taillard, 1991); and WalkSAT (Selman et al., 1994) on hard random 3-SAT problems (Hoos and Stützle, 1999).

The objective of this paper is to determine if the solution times for GRASP also have this property, i.e., they fit a two parameter exponential distribution. To do this, we consider five GRASPs that have been reported in the literature and for which we have source code:

1. maximum independent set (Feo et al., 1994; Resende et al., 1998);
2. quadratic assignment problem (Li et al., 1994; Resende et al., 1996);
3. graph planarization (Resende and Ribeiro, 1997; Ribeiro and Resende, 1999);
4. maximum weighted satisfiability (Resende et al., 1997; 2000);
5. maximum covering (Resende, 1998).

For each GRASP, we selected four test problems from the literature. For each of these instances, we determined three solution target values spread out between minimum and maximum values produced by GRASP. For each target value, we measured running times to find a solution at least as good as the target and studied these distributions.

The remainder of this paper is organized as follows. In Section 6.2, we give a brief overview of each of the five GRASPs used in this study. The experimental design is described in Section 6.3. The experimental results are reported in Section 6.4. In Section 6.5, we make concluding remarks.

6.2 FIVE GRASP IMPLEMENTATIONS

In this section, we briefly describe the five GRASPs used in the experiments. For each GRASP, we define the combinatorial optimization problem it solves, the construction phase and the local search phase.

6.2.1 GRASP for maximum independent set

A GRASP for maximum independent set was introduced by Feo et al. (1994). Fortran subroutines that implement this GRASP are found in Resende et al. (1998).

6.2.1.1 Problem definition. Let $G = (V,E)$ be an undirected graph with vertex set V and edge set E. Vertices $u,v \in V$ are *nonadjacent* if $(u,v) \notin E$. A subset of the vertices $S \subseteq V$ is *independent* if all vertices in S are pairwise nonadjacent. In the *maximum independent set problem* one wants to find an independent set having the largest cardinality.

6.2.1.2 Construction phase. The algorithm initializes a working graph $\tilde{G} = (\tilde{V},\tilde{E})$ to be the original graph, and sets the independent set S empty. The independent set is built up one vertex at a time. The greedy function that guides the construction is vertex degree with respect to the working graph. It selects among the working graph vertices, the one with minimum degree and places that vertex in the independent set. The greedy function is adaptive, since it changes with the selection of each independent vertex.

Let $\underline{d}$ and $\overline{d}$ be, respectively, the minimum and maximum degrees over all working vertices, i.e.

$$\underline{d} = \min_{v \in \tilde{V}} \{d(v, \tilde{G})\} \text{ and } \overline{d} = \max_{v \in \tilde{V}} \{d(v, \tilde{G})\},$$

where $d(v, \tilde{G})$ is the degree of vertex v with respect to $\tilde{G}$. The *restricted candidate list* (RCL) is the set of vertices

$$\text{RCL} = \{v \in \tilde{V} \mid d(v, \tilde{G}) \leq \underline{d} + \alpha(\overline{d} - \underline{d})\},$$

where the parameter α controls the size of the RCL and is such that $0 \leq \alpha \leq 1$. The vertex selection in the GRASP construction phase is random, restricted to vertices in the RCL.

6.2.1.3 Local search phase. A $(2,1)$-exchange local search heuristic for the maximum independent set problem seeks a larger independent set by removing a single vertex x from the independent set S and replacing it by two nonadjacent vertices u and v, such that u and v are not adjacent to any vertex in $S \setminus \{x\}$. If such an exchange is found, the procedure is recursively applied on the new larger independent set. A locally optimal solution is detected when no further exchange is possible.

6.2.2 GRASP for quadratic assignment

Li et al. (1994) introduce a GRASP for the quadratic assignment problem (QAP) and describe Fortran subroutines for this GRASP in (Resende et al., 1996). A specialization of this GRASP for sparse QAPs together with Fortran subroutines are presented in (Pardalos et al., 1997). A parallel version of this GRASP can be found in (Pardalos et al., 1995). Improvements to the construction and local search phases are described in (Fleurent and Glover, 1999; Rangel et al., 1999; 1998).

6.2.2.1 Problem definition. Given a set $\mathcal{N} = \{1, 2, \ldots, n\}$ and $n \times n$ matrices $F = (f_{ij})$ and $D = (d_{kl})$, the quadratic assignment problem (QAP) can be stated as follows:

$$\min_{p \in \Pi_{\mathcal{N}}} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{p(i)p(j)},$$

where $\Pi_{\mathcal{N}}$ is the set of all permutations of $\mathcal{N}$.

6.2.2.2 Construction phase. The construction phase has two stages. In stage 1, two assignments are produced, i.e. facility i is assigned to site k and facility j is assigned to site l. The idea is to assign facilities with high interaction (having high f_{ij} values) to nearby sites (site pairs with low d_{kl} values). To do this, the procedure sorts inter-site distances in increasing order and inter-facility flows in decreasing order. Let $d_{k_1,l_1} \leq d_{k_2,l_2} \leq \cdots \leq d_{k_p,l_p}$ and $f_{i_1,j_1} \geq f_{i_2,j_2} \geq \cdots \geq f_{i_p,j_p}$ be the sorted values, where $p = n^2 - n$. The products $d_{k_1,l_1} \cdot f_{i_1,j_1}, d_{k_2,l_2} \cdot f_{i_2,j_2}, \ldots, d_{k_p,l_p} \cdot f_{i_p,j_p}$ are then sorted in increasing order. Among the smallest $d_{kl} \cdot f_{ij}$ products, one (corresponding to the pair of stage 1 assignments) is selected at random. Sorting all of the $p = n^2 - n$ distances and flows is inefficient and offers little benefit. Instead, only the best $n_\beta = \beta p$ values

are sorted, where β is a parameter such that $0 < \beta \leq 1$. Among these n_β pairs of assignments, a pair is selected at random from the set of αn_β assignments having the smallest $d_{kl} \cdot f_{ij}$ products, where α is such that $0 < \alpha \leq 1$.

In stage 2 of the construction phase, the remaining $n-2$ facility-site assignments are made sequentially. The idea is to favor assignments that have small interaction cost with the set of previously-made assignments. Let Γ be the set of q assignments at a given point in the construction phase, i.e. $\Gamma = \{(i_1,k_1),(i_2,k_2),\ldots,(i_q,k_q)\}$. The cost of assigning facility j to site l, with respect to the already-made assignments, is defined to be

$$c_{jl} = \sum_{(i,k)\in\Gamma} f_{ij} d_{kl}.$$

All costs of unassigned facility-site pairs (j,l) are sorted in increasing order. Of the pairs having the least $\alpha \cdot |\Gamma|$ costs, one is selected at random and is added to the set Γ. The procedure is repeated until $n-1$ assignments are made. The remaining facility is then assigned to the remaining site.

6.2.2.3 Local search phase. In the local search phase of this GRASP, a 2-exchange neighborhood search is conducted on the constructed solution. There, all possible 2-swaps of facility-locations are considered. If a swap improves the cost of the assignment, it is accepted. The procedure continues until no swap improves the solution value.

6.2.3 GRASP for graph planarization

A GRASP for graph planarization was introduced in Resende and Ribeiro (1997). Fortran subroutines for their algorithm are described in (Ribeiro and Resende, 1999).

6.2.3.1 Problem definition. A graph is said to be *planar* if it can be drawn on the plane in such a way that no two of its edges cross. Given a graph $G = (V,E)$ with vertex set V and edge set E, the objective of *graph planarization* is to find a minimum cardinality subset of edges $F \subseteq E$ such that the graph $G' = (V, E \setminus F)$, resulting from the removal of the edges in F from G, is planar. This problem is also known as the *maximum planar subgraph* problem.

6.2.3.2 Two phase heuristic. The GRASP described in this section is based on the separation of the computation into two phases (Goldschmidt and Takvorian, 1994). The first phase consists in devising a linear permutation of the nodes of the input graph, followed by placing them along a line. The second phase determines two sets of edges that may be represented without crossings above and below that line, respectively.

The GRASP construction and local search are applied to the first phase, where a linear permutation of the nodes is determined.

6.2.3.3 Construction phase. After the first k nodes of the permutation have been determined, say $v_1, v_2, \cdots, v_k$, the next node v_{k+1} is selected at random from the nodes adjacent to v_k in G having the lowest degrees in the subgraph G_k of G induced

by $V \setminus \{v_1, v_2, \cdots, v_k\}$. If there is no node of G_k adjacent to v_k in G, then v_{k+1} is selected at random from a set of low degree nodes in G_k.

6.2.3.4 Local search phase. The local search phase explores the neighborhood of the current permutation by swapping the positions of two nodes at a time, attempting to reduce the number of possible edge crossings.

6.2.3.5 Post-optimization. Each iteration of this GRASP produces three edge sets: $\mathcal{B}$ (blue edges, which are drawn above the line), $\mathcal{R}$ (red edges, which are drawn below the line), and $\mathcal{P}$ (the remaining edges, which are referred to as the *pale* edges). By construction, $\mathcal{B}$, $\mathcal{R}$, and $\mathcal{P}$ are such that no red or pale edge can be colored blue. Likewise, pale edges cannot be colored red. However, if there exists a pale edge $p \in \mathcal{P}$ such that all blue edges that cross with p (let $\hat{\mathcal{B}}_p \subseteq \mathcal{B}$ be the set of those blue edges) do not cross with any red edge $r \in \mathcal{R}$, then all blue edges $b \in \hat{\mathcal{B}}_p$ can be colored red and p can be colored blue. In case this reassignment of colors is possible, then the size of the planar subgraph is increased by one edge. This post-optimization procedure is incorporated at the end of each GRASP iteration.

6.2.4 GRASP for MAX-SAT

A GRASP for satisfiability was first proposed in Resende and Feo (1996). This GRASP was generalized to handle MAX-SAT problems by Resende et al. (1997). A parallel version of this algorithm is described in (Pardalos et al., 1996) and Fortran subroutines are presented in (Resende et al., 2000).

6.2.4.1 Problem definition. Let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ be m clauses, involving n Boolean variables x_1, x_2, …, x_n, which can take on only the values `true` or `false` (1 or 0). In addition, for each clause $\mathcal{C}_i$, there is an associated nonnegative weight w_i. Define clause i to be

$$\mathcal{C}_i = \bigvee_{j=1}^{n_i} l_{ij},$$

where n_i is the number of literals in clause $\mathcal{C}_i$, and literal $l_{ij} \in \{x_i, \bar{x}_i \mid i = 1, \ldots, n\}$. A clause is said to be *satisfied* if it evaluates to `true`. In the weighted *Maximum Satisfiability Problem* (MAX-SAT), one is to determine the assignment of truth values to the n variables that maximizes the sum of the weights of the satisfied clauses.

6.2.4.2 Construction phase. A feasible solution to a MAX-SAT instance is described by $x \in \{0,1\}^n$. Let $w(x)$ is the sum of the weights of the clauses satisfied by x. The construction phase solution is built, one element at a time, guided by a greedy function and randomization. Since in the MAX-SAT problem there are n variables to be assigned, each construction phase consists of n iterations.

The idea behind the greedy function is to maximize the total weight of yet-unsatisfied clauses that become satisfied after the assignment of each construction phase iteration. For $i \in N$, let Γ_i^+ be the set of unassigned clauses that would become satisfied if variable x_i were to be set to `true`. Likewise, let Γ_i^- be the set of unassigned clauses that

would become satisfied if variable x_i were to be set to `false`. Define

$$\gamma_i^+ = \sum_{j \in \Gamma_i^+} w_j \text{ and } \gamma_i^- = \sum_{j \in \Gamma_i^-} w_j.$$

The greedy choice is to select the variable x_k with the largest γ_k^+ or γ_k^- value and set it to the corresponding truth value. If $\gamma_k^+ > \gamma_k^-$, then the assignment $x_k = 1$ is made, else $x_k = 0$. Note that with every assignment made, the sets Γ_i^+ and Γ_i^- change for all i such that x_i is not assigned a truth value, to reflect the new assignment. This consequently changes the values of γ_i^+ and γ_i^-, characterizing the adaptive component of the heuristic.

Let

$$\gamma^* = \max\{\gamma_i^+, \gamma_i^- \mid x_i \text{ yet unassigned}\}$$

and

$$\gamma_* = \min\{\gamma_i^+, \gamma_i^- \mid x_i \text{ yet unassigned}\},$$

and let α ($0 \leq \alpha \leq 1$) be the restricted candidate parameter. A new value for α is selected, at random, at each iteration, from the uniform distribution $U[0,1]$. A candidate $x_i =$ `true` is inserted into the RCL if $\gamma_i^+ \geq \gamma_* + \alpha \cdot (\gamma^* - \gamma_*)$. Likewise, a candidate $x_i =$ `false` is inserted if $\gamma_i^- \geq \gamma_* + \alpha \cdot (\gamma^* - \gamma_*)$.

6.2.4.3 Local search phase. To define the local search procedure, some preliminary definitions have to be made. Given a truth assignment $x \in \{0,1\}^n$, define the *1-flip neighborhood* $N(x)$ to be the set of all vectors $y \in \{0,1\}^n$ such that $\| x - y \|_2 = 1$. If x is interpreted as a vertex of the n-dimensional unit hypercube, then its neighborhood consists of the n vertices adjacent to x. If we denote by $w(x)$ the total weight of the clauses satisfied by the truth assignment x, then the truth assignment x is a *local maximum* if and only if $w(x) \geq w(y)$, for all $y \in N(x)$. Starting with a truth assignment x, the local search finds the local maximum y in $N(x)$. If $y \neq x$, it sets $x = y$. This process is repeated until no further improvement is possible.

6.2.5 GRASP for maximum covering

A GRASP for the maximum covering problem is described in Resende (1998).

6.2.5.1 Problem definition. The maximum covering problem can be stated as: Let $J = \{1,2,\ldots,n\}$ denote the set of n potential facility locations. Define n finite sets $P_1, P_2, \ldots, P_n$, each corresponding to a potential facility location, such that $I = \cup_{j \in J} P_j = \{1,2,\ldots,m\}$ is the set of the m demand points that can be covered by the n potential facilities. With each demand point $i \in I$, we associate a weight $w_i \geq 0$. A *cover* $J^* \subseteq J$ covers the demand points in set $I^* = \cup_{j \in J^*} P_j$ and has an associated weight $w(J^*) = \sum_{i \in I^*} w_i$. Given the number $p > 0$ of facilities to be placed, we wish to find the set $J^* \subseteq J$ that maximizes $w(J^*)$, subject to the constraint that $|J^*| = p$.

6.2.5.2 Construction phase. Since in the maximum covering problem there are p facility locations to be chosen, each construction phase consists of p iterations, with one location chosen per iteration.

To define a restricted candidate list, we rank the yet unchosen facility locations according to an adaptive greedy function. Let J^* denote the set (initially empty) of chosen facility locations being built in the construction phase. At any construction phase iteration, let Γ_j be the set of additional uncovered demand points that would become covered if facility location j (for $j \in J \setminus J^*$) were to be added to J^*. Define the *greedy function*

$$\gamma_j = \sum_{i \in \Gamma_j} w_i$$

to be the incremental weight covered by the choice of facility location $j \in J \setminus J^*$. The greedy choice is to select the facility location k having the largest γ_k value. Note that with every selection made, the sets Γ_j, for all yet unchosen facility location indices $j \in J \setminus J^*$, change to reflect the new selection. This consequently changes the values of the greedy function γ_j, characterizing the adaptive component of the heuristic.

Let

$$\gamma^* = \max\{\gamma_j \mid \text{facility location } j \text{ is still unselected, i.e. } j \in J \setminus J^*\}$$

and α be the restricted candidate parameter ($0 \leq \alpha \leq 1$). We say a facility location j is a *potential candidate*, and is added to the RCL, if $\gamma_j \geq \alpha \times \gamma^*$. A location is selected at random from the RCL and is added to the solution.

6.2.5.3 Local search phase. Two solutions (sets of facility locations) J^1 and J^2 are said to be neighbors in the 2-exchange neighborhood if they differ by exactly one element, i.e. $| J^1 \cap \Delta J | = | J^2 \cap \Delta J | = 1$, where $\Delta J = (J^1 \cup J^2) \setminus (J^1 \cap J^2)$. The local search starts with a set J^* of p facility locations, and at each iteration attempts to find a pair of locations $s \in J^*$ and $t \in J \setminus J^*$ such that $w(J^* \setminus \{s\} \cup \{t\}) > w(J^*)$. If such a pair exists, then location s is replaced by location t in J^*. A solution is locally optimal with respect to this neighborhood if there exists no pairwise exchange that increases the total weight of J^*.

6.3 EXPERIMENTAL DESIGN

In this section we describe the experimental design. We analyze five GRASPs that have appeared in the literature and for which source code is available. For each of these algorithms, we select four test problems to study the probability distribution of solution time. The hypothesis of this paper is that CPU times fit a two parameter exponential distribution. We measure the CPU time to find an objective function value at least as good as a given target value. This is done for three different target values for each test problem. These values are spread out between a value far from the optimal and the best value produced by GRASP. Each GRASP is run $n = 200$ times for all instance/target combinations. For each of the 200 runs of each combination, the random number generator is initialized with a distinct seed and therefore the runs are independent. To compare the empirical and the theoretical distributions, we follow a standard graphical methodology for data analysis (Chambers et al., 1983). In the remainder of this section we describe this methodology.

For each instance/target pair, the running times are sorted in increasing order. We associate with the i-th sorted running time (t_i) a probability $p_i = (i - \frac{1}{2})/n$, and plot

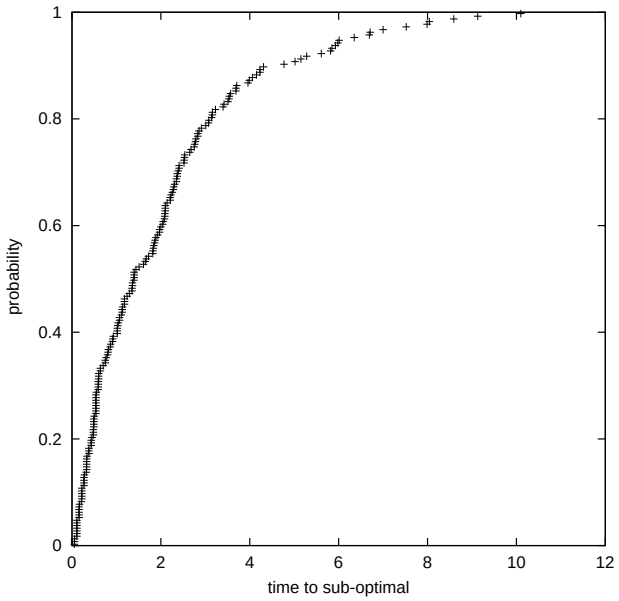


Figure 6.1 Cumulative probability distribution plot of measured data.

the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, n$. We comment on this choice of p_i later in this section. Figure 6.1 illustrates this cumulative probability distribution plot for one of the instance/target pairs.

To estimate the parameters of the two-parameter exponential distribution, we first draw the theoretical quantile-quantile plot (or Q-Q plot) for the data. To describe Q-Q plots, we recall that the cumulative distribution function for the two-parameter exponential distribution is given by

$$F(t) = 1 - e^{-(t-\mu)/\lambda},$$

where λ is the mean of the distribution data (and indicates the spread of the data) and μ is the shift of the distribution with respect to the ordinate axis.

For each value p_i, $i = 1, \ldots, n$, we associate a p_i-quantile $Qt(p_i)$ of the theoretical distribution. For each p_i-quantile we have, by definition, that

$$F((Qt(p_i)) = p_i.$$

Hence, $Qt(p_i) = F^{-1}(p_i)$ and therefore, for the two-parameter exponential distribution, we have

$$Qt(p_i) = -\lambda \ln(1 - p_i) + \mu.$$

The quantiles of the data of an empirical distribution are simply the (sorted) raw data. Note that if we were to use $p_i = i/n$, for $i = 1, \ldots, n$, then $Qt(p_n)$ would be undefined.

A theoretical quantile-quantile plot (or theoretical Q-Q plot) is obtained by plotting the quantiles of the data of an empirical distribution against the quantiles of a theoretical distribution. This involves three steps. First, the data (in our case, the measured

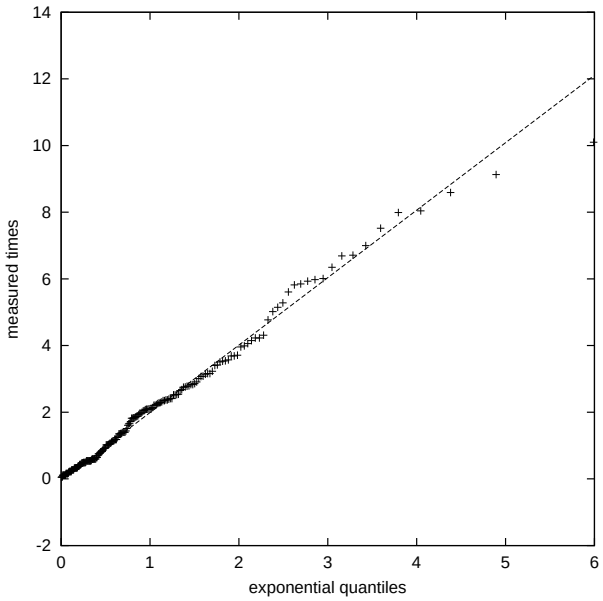


Figure 6.2 Q-Q plot showing fitted line.

times) are sorted in ascending order. Second, the quantiles of the theoretical exponential distribution are obtained. Finally, a plot of the data against the theoretical quantiles is made.

In a situation where the theoretical distribution is a close approximation of the empirical distribution, the points in the Q-Q plot will have a nearly straight configuration. If the parameters λ and μ of the theoretical distribution that best fits the measured data could be estimated a priori, the points in a Q-Q plot would tend to follow the line $x = y$. Alternatively, in a plot of the data against a two-parameter exponential distribution with $\lambda = 1$ and $\mu = 0$, the points would tend to follow the line $y = \hat{\lambda}x + \hat{\mu}$. This means that a single theoretical Q-Q plot compares a set of data not just to one theoretical distribution, but simultaneously to a whole family of distributions. Consequently, parameters λ and μ of the two-parameter exponential distribution can be estimated, respectively, by the slope $\hat{\lambda}$ and intercept $\hat{\mu}$ of the line depicted in the Q-Q plot.

The Q-Q plot shown in Figure 6.2 is obtained by plotting the measured times in the ordinate against the quantiles of a two-parameter exponential distribution with $\lambda = 1$ and $\mu = 0$ in the abscissa, given by $-\ln(1 - p_i)$ for $i = 1, \ldots, n$. To avoid possible distortions caused by outliers, we do not estimate the distribution mean with the data mean or by linear regression on the points of the Q-Q plot. Instead, we estimate the slope $\hat{\lambda}$ of line $y = \lambda x + \mu$ using the upper quartile q_u and lower quartile q_l of the data. The upper and lower quartiles are, respectively, the $Q(\frac{1}{4})$ and $Q(\frac{3}{4})$ quantiles, respectively. We take

$$\hat{\lambda} = (z_u - z_l)/(q_u - q_l)$$

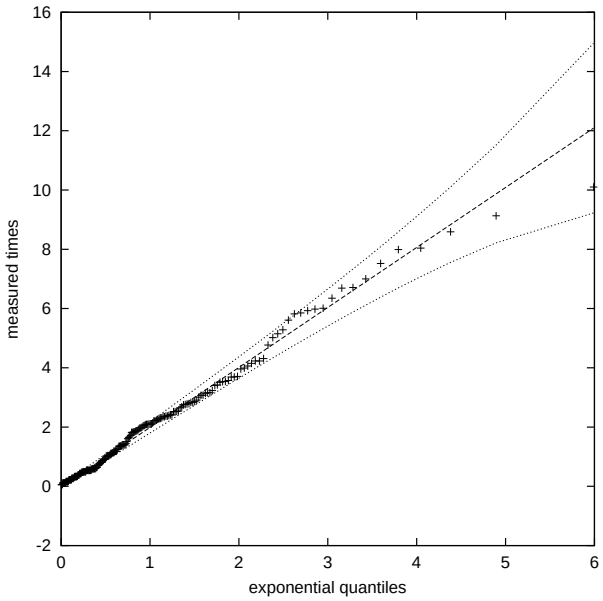


Figure 6.3 Q-Q plot with variability information.

as an estimate of the slope, where z_u and z_l are the u-th and l-th points of the ordered measured times, respectively. This informal estimation of the distribution of the measured data mean is robust since it will not be distorted by a few outliers (Chambers et al., 1983).

To analyze the straightness of the Q-Q plots, we superimpose them with variability information. For each plotted point, we show plus and minus one standard deviation in the vertical direction from the line fitted to the plot. An estimate of the standard deviation for point z_i, $i = 1, \ldots, n$, of the Q-Q plot is

$$\hat{\sigma} = \hat{\lambda}\sqrt{\frac{p_i}{(1-p_i)n}}.$$

Figure 6.3 shows an example of a Q-Q plot with superimposed variability information.

When observing a theoretical quantile-quantile plot with superimposed standard deviation information, one should avoid turning such information into a formal test. One important fact that must be kept in mind is that the natural variability of the data generates departures from the straightness, even if the model of the distribution is valid. The most important reason for portraying standard deviation is that it gives us a sense of the relative variability of the points in the different regions of the plot. However, since one is trying to make simultaneous inferences from many individual inferences, it is difficult to use standard deviations to judge departures from the reference distribution. For example, the probability that a particular point deviates from the reference line by more than two standard deviations is small. But the probability that at least one of the data points deviates from the line by two standard deviations is

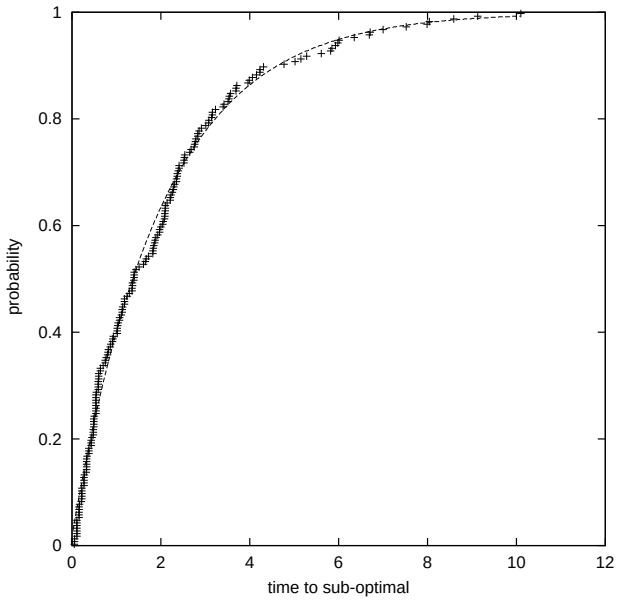


Figure 6.4 Superimposed empirical and theoretical distributions.

probably much greater. In order statistics, this is made more difficult by the high correlation that exists between neighboring points. If one plotted point deviates by more than one standard deviation, there is a good chance that a whole bunch of them will too. Another point to keep in mind is that standard deviations vary substantially in the Q-Q plot, as can be observed in the Q-Q plot in Figure 6.3 that the standard deviations of the points near the high end are substantially larger then the standard deviation of the other end.

Once the two parameters of the distribution are estimated, a superimposed plot of the empirical and theoretical distributions can be made. Figure 6.4 shows this plot corresponding to the Q-Q plot in Figure 6.2.

6.4 COMPUTATIONAL RESULTS

In this section, we present the computational results. We describe the computer environment used to conduct the experiments, the instances selected for each of the five GRASPs, and present for each GRASP/instance/target triplet its Q-Q plot with variability information, the two estimated parameters, and superimposed plots of the empirical and theoretical distributions.

6.4.1 Computer environment

The experiments were done on an SGI Challenge computer (28 196 MHz MIPS R10000 processors) with 7.6 Gb of memory. Each run used a single processor.

Table 6.2 Maximum independent set test problems with best known solutions (bks), target values, and parameter estimates.

			estimates	
problem	bks	target	$\hat{\mu}$	$\hat{\lambda}$
270002	15	13	0.015	0.146
		14	-0.018	1.612
		15	25.560	291.715
270003	15	13	0.068	0.147
		14	0.142	5.797
		15	1.849	223.491
270004	15	13	0.034	0.092
		14	-0.035	2.024
		15	1.339	30.248
270006	15	13	0.021	0.137
		14	-0.013	1.383
		15	38.909	516.965

The algorithms were coded in Fortran and were modified minimally to produce the data needed for the experiments. CPU times were measured with the system function `etime`. The codes were compiled with the SGI MIPSpro F77 compiler. The GRASPs for maximum independent set and quadratic assignment were compiled using flags `-Ofast -u` and the GRASP for maximum satisfiability was compiled using flags `-Ofast -static`. The GRASP for maximum covering was compiled using flags `-O3 -r4 -64` and the GRASP for planarization was compiled using flags `-O3 -static`.

6.4.2 Test problems

The test problem names, their best known solutions, and respective target values are shown in Tables 6.2– 6.6.

The four problems used to study the GRASP for maximum independent set were chosen from a much studied class of random graphs (Bollobás, 1985), denoted by $G_{n,p}$. Such graphs have n nodes and each edge (i, j), $i, j = 1, \ldots, n$, $i \neq j$, exists with probability p. The experiment consisted in running the algorithm on four random instances of $G_{n,p}$ $(n = 1000, p = 0.5)$ generated with the Fortran code in the distribution (Resende et al., 1998).

For the GRASP for QAP, test problems `chr25a`, `kra30b`, `sko42`, and `tho40`, were chosen from the suite of QAP test programs QAPLIB (Burkard et al., 1991). The problems are pure quadratic assignment problems that have at least one symmetric distance or flow matrix. Their dimensions (n) range from 25 to 42.

The GRASP for graph planarization was tested for four problems (`g17`, `tg100.10`, `rg100.1`, and `rg150.1`) chosen from a set of 75 test problems described in the liter-

Table 6.3 Quadratic assignment problem test problems with best known solutions, target values, and parameter estimates.

			estimates	
problem	bks	target	$\hat{\mu}$	$\hat{\lambda}$
chr25a	3796	5023	0.016	0.221
		4721	0.012	1.147
		4418	0.460	8.401
kra30b	91420	94675	0.021	0.076
		93590	0.024	0.261
		92505	-0.041	2.045
sko42	15812	16389	0.051	0.049
		16222	0.044	0.244
		16055	0.173	1.955
tho40	240516	247160	0.105	0.419
		245396	0.255	2.163
		243632	3.397	19.413

Table 6.4 Graph planarization test problems with best known solutions, target values, and parameter estimates.

			estimates	
problem	bks	target	$\hat{\mu}$	$\hat{\lambda}$
g17	236	222	2.564	8.723
		227	-0.738	72.498
		231	-19.991	763.081
tg100.10	277	215	0.575	0.690
		226	0.747	5.120
		236	-10.540	181.038
rg100.1	162	154	0.135	0.563
		157	0.062	3.983
		159	-0.148	25.954
rg150.1	231	215	0.531	0.722
		220	0.368	6.961
		225	3.495	286.668

ature (Cimikowski, 1995; Goldschmidt and Takvorian, 1994). The dimensions of the selected problems range from 100 to 150 vertices and 261 to 742 edges.

The test problems for the MAX-SAT problem were chosen among the instances reported in Resende et al. (1997). Problems jnh11 and jnh12 have 100 variables and 800 clauses, problem jnh212 has 100 variables and 850 clauses, and problem jnh306 has 100 variables and 900 clauses.

Table 6.5 Maximum satisfiability test problems with best known solutions, target values, and parameter estimates.

maximum satisfiability				
			estimates	
problem	bks	target	$\hat{\mu}$	$\hat{\lambda}$
jnh11	420753	418851	0.061	0.136
		419485	0.063	0.876
		420119	0.860	24.903
jnh12	420925	417664	0.053	0.046
		418751	0.044	0.233
		419838	0.064	2.797
jnh212	394238	393145	0.033	0.707
		393506	-0.226	4.148
		393866	-0.261	46.058
jnh306	444838	441720	0.059	0.058
		442759	0.062	0.219
		443798	-0.198	3.509

Table 6.6 Maximum covering test problems with best known solutions, target values, and parameter estimates.

maximum covering				
			estimates	
problem	bks	target	$\hat{\mu}$	$\hat{\lambda}$
r24-500	33343542	33330441	-2.257	109.827
		33334808	11.960	229.850
		33339175	2.273	669.501
r25-250	20606926	20567005	1.042	3.319
		20580312	0.716	14.555
		20593619	4.279	101.40
r54-100	39684669	39332719	6.272	42.187
		39450036	17.803	272.29
		39567352	73.320	3978.427
r55-100	39504338	39037219	6.917	9.063
		39192925	3.190	37.672
		39348631	-10.888	279.470

The test problems for the GRASP for maximum covering (r24-500, r25-250, r54-100, and r55-100) were generated randomly using the generator described in Resende (1998). All instances have 1000 potential location sites and demand points varying from 7425 to 9996. The number of facilities to be located varies from 100 to 500.

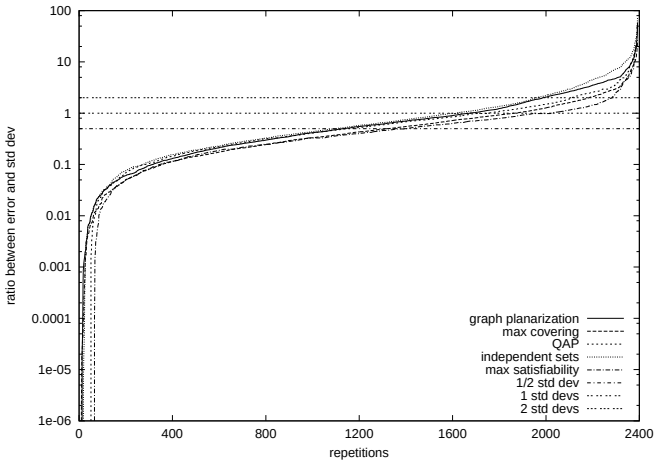


Figure 6.5 Ratio of error and standard deviation for all repetitions – All instances.

6.4.3 Generating the data points

200 independent runs of each GRASP were done for each instance/target pair. In each run, the GRASP was halted after a solution at least as good as the target was found and the total CPU time (in seconds) was recorded. With the 200 data points generated, a Q-Q plot with variability information and a superimposed plot of the empirical and theoretical distributions were made.

6.4.4 Q-Q plots and theoretical distributions

In this subsection, we present Q-Q plots with variability information with the corresponding plots of superimposed empirical and theoretical distributions.

Each Q-Q plot figure is composed of 12 Q-Q plots, one for each instance/target pair. Likewise, each superimposed empirical and theoretical distribution figure has 12 plots, one for each instance/target pair. Each plot is made up of 200 data points. Each figure has four rows of plots, each corresponding to one of the four instances. For each instance, three increasingly difficult target values are used. In each row of the figure, the difficulty of finding a solution with a given target value increases from left to right.

The estimated parameters for all GRASPs are shown in Tables 6.2– 6.6. Parameter $\hat{\lambda}$ is the estimated mean time to target solution and parameter $\hat{\mu}$ is the estimated minimum time to target solution. Since the minimum time to target solution is the time corresponding to one GRASP iterations, $\hat{\mu}$ is an estimate of one GRASP iteration.

Figures A.1 and A.2 show, respectively, the Q-Q plots and superimposed empirical and theoretical distributions for the maximum independent set instances. For the quadratic assignment problem instances, the Q-Q plots and superimposed empirical and theoretical distributions are shown, respectively, in Figures A.3 and A.4. Figures A.5 and A.6 depict, respectively, the Q-Q plots and superimposed empirical and theoretical distributions for the graph planarization instances. For the maximum satis-

fiability instances, the Q-Q plots and superimposed empirical and theoretical distributions are depicted, respectively, in Figures A.7 and A.8. Figures A.9 and A.10 show, respectively, the Q-Q plots and superimposed empirical and theoretical distributions for the maximum covering instances.

We make the following observations regarding the experiments.

12,000 independent runs were carried out, each finding a solution at least as good as its respective target.

In general, there was no large or systematic departure from straightness in the Q-Q plots. However, it is well known in statistics that the samples of real data are often contaminated by a small fraction of abnormal observations that will lie outside the typical range of data. This can be observed in the experiments reported here.

Straightness generally increased with problem difficulty, i.e. as the target value approached the optimal, the fit of solution time to the theoretical distribution improved, implying therefore that the computed parameters were good estimates. This occurs because the distribution of number of GRASP iterations until target solution is more spread out. In some of the easier instances, many runs took few iterations. We further discuss this issue later in this section.

The points in each Q-Q plot can be regarded as order statistics. Due to the high correlation that exists between neighboring order statistics, the probability that a particular point deviates from the line by more than two standard deviations is small. However, as commented in Section 6.3, the probability that at least one of the points deviates from the line by two standard deviations is undoubtedly much greater. Figures 6.5 and 6.6 plot the ratios of deviation from the fitted line to one standard deviation, for all 2400 instances of each GRASP and all 800 harder instances of each GRASP, respectively. The harder instances correspond to those in the rightmost column of the Q-Q plots and superimposed plots of the empirical and theoretical distributions. The ratios are sorted in increasing order. About 75% of all points in the Q-Q plots fall within one standard deviation of the fitted line and about 88% fall within two standard deviations. When limited only to hard instance/target pairs, then about 80% of the all points in the Q-Q plots fall within one standard deviation of the fitted line and about 93% fall within two standard deviations.

As can be seen in the Q-Q plots, not many points associated with large CPU times fall outside the one standard deviation bounds. Hence, most of the points that fall outside the two standard deviation bounds are points associated with small CPU times which in turn have small standard deviations. In fact, if we only consider the largest 180 (of 200) CPU times for each instance/target pair, we observe that 93% of all points in the Q-Q plots fall within two standard deviations of the fitted line. Restricting our sample to only hard instance/target plots, then 98% of the points fall within the bounds.

A quantile-quantile plot with horizontal segments, as observed in Figures A.1 and A.5, is common in practice (Chambers et al., 1983). This discrete granularity may mean that the data were rounded at some earlier stage before being plotted. In our experiments, this is observed only for the easiest problem/target pairs. It occurs because in many runs, the target solution was found in the same GRASP iteration. In these examples, the i-th horizontal segment depicts the solution times found in the i-th GRASP iteration. Although this is a departure from normality, if these repetitions were elim-

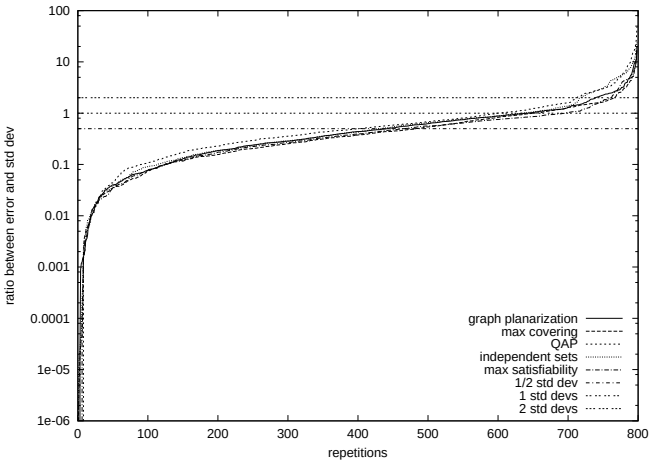


Figure 6.6 Ratio of error and standard deviation for repetitions – Only harder instances.

inated and each segment turned into a single point, represented by its median, nearly straight lines would be observed for all these plots.

6.5 CONCLUDING REMARKS

Though it is clear that distributing the GRASP iterations evenly among parallel processors achieves linear speedup for total time (to run all GRASP iterations), it is less clear why linear speedup in time to a given target value is frequently observed. If time to a target solution value fits a two-parameter exponential distribution, then the probability of finding a solution of a given value in time ρt with a sequential process is equal to the probability of finding a solution at least as good as that given value in time t with ρ independent parallel processes. Hence, linear speedup would be observed.

In this paper, we reported on an empirical investigation of the distribution of solution time to a target value. 12,000 GRASP runs were done in the experiment. To analyze the observations, we used a standard graphical methodology for data analysis. Q-Q plots with variability information were used to determine if the empirical distribution fits its theoretical counterpart. The estimated parameters of the theoretical distribution were derived from the Q-Q plots.

The main conclusion from the experiments is that time to target value indeed fits well a two-parameter exponential distribution. The fit tends to improve as the difficulty to find a solution of a given target value increases.

Though this study was limited to five distinct GRASPs, we believe that this characteristic is present in any GRASP implemented in a straightforward manner. It should be noted that tricks commonly used to speedup a sequential GRASP, such as hash tables to avoid repetition of local search from identical starting solutions, can make speedup in time to target solution be sublinear in the number of processors. An example of this can be seen in Martins et al. (2000), where the use of a hash table improves

the speed of a sequential GRASP in instances in which the construction phase generates many identical solutions and the parallel GRASP repeats many of these unnecessary local searches.

Acknowledgment

R.M. Aiex was supported by the Brazilian Council for Scientific and Technological Development (CNPq) and was done while this author was visiting AT&T Labs Research.

Bibliography

R. Battiti and G. Tecchiolli. Parallel biased search for combinatorial optimization: Genetic algorithms and TABU. *Microprocessors and Microsystems*, 16:351–367, 1992.

B. Bollobás. *Random graphs*. Academic Press, 1985.

R. Burkard, S. Karisch, and F. Rendl. QAPLIB – A quadratic assignment problem library. *European Journal of Operations Research*, 55:115–119, 1991.

J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. *Graphical Methods for Data Analysis*. Chapman & Hall, 1983.

R.J. Cimikowski. An analysis of heuristics for the maximum planar subgraph problem. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 322–331, 1995.

N. Dodd. Slow annealing versus multiple fast annealing runs: An empirical investigation. *Parallel Computing*, 16:269–272, 1990.

H.M.M. Ten Eikelder, M.G.A. Verhoeven, T.W.M. Vossen, and E.H.L. Aarts. A probabilistic analysis of local search. In I.H. Osman and J.P. Kelly, editors, *Metaheuristics: Theory & applications*, pages 605–618. Kluwer Academic Publishers, 1996.

T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.

P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. Technical report, AT&T Labs Research, Florham Park, NJ 07733, 2000.

C. Fleurent and F. Glover. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing*, 11: 198–204, 1999.

O. Goldschmidt and A. Takvorian. An efficient graph planarization two-phase heuristic. *Networks*, 24:69–73, 1994.

J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.

H.H. Hoos and T. Stützle. Towards a characterisation of the behaviour of stochastic local search algorithms for sat. *Artificial Intelligence*, 112:213–232, 1999.

Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994.

S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P.M. Pardalos. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization*, 2000. To appear.

S.L. Martins, C.C. Ribeiro, and M.C. Souza. A parallel GRASP for the Steiner problem in graphs. In A. Ferreira and J. Rolim, editors, *Proceedings of IRREGULAR'98 – 5th International Symposium on Solving Irregularly Structured Problems in Parallel*, volume 1457 of *Lecture Notes in Computer Science*, pages 285–297. Springer-Verlag, 1998.

R.A. Murphey, P.M. Pardalos, and L.S. Pitsoulis. A parallel GRASP for the data association multidimensional assignment problem. In P.M. Pardalos, editor, *Parallel processing of discrete problems*, volume 106 of *The IMA Volumes in Mathematics and Its Applications*, pages 159–180. Springer-Verlag, 1998.

L.J. Osborne and B.E. Gillett. A comparison of two simulated annealing algorithms applied to the directed Steiner problem on networks. *ORSA J. on Computing*, 3: 213–225, 1991.

P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP implementation for the quadratic assignment problem. In A. Ferreira and J. Rolim, editors, *Parallel Algorithms for Irregularly Structured Problems – Irregular'94*, pages 111–130. Kluwer Academic Publishers, 1995.

P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP for MAX-SAT problems. *Lecture Notes in Computer Science*, 1184:575–585, 1996.

P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. Algorithm 769: Fortran subroutines for approximate solution of sparse quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software*, 23:196–208, 1997.

M.C. Rangel, N.M.M. Abreu, and P.O. Boaventura Netto. GRASP in the QAP: An acceptance bound for initial solution. In *Proc. of the Third Metaheuristics International Conference*, pages 381–386, July 1999.

M.C. Rangel, N.M.M. de Abreu, P.O. Boaventura Netto, and M.C.S. Boeres. A modified local search for GRASP in the quadratic assignment problem. Technical report, Production Engineering Program, COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, RJ Brazil, 1998.

M.G.C. Resende. Computing approximate solutions of the maximum covering problem using GRASP. *J. of Heuristics*, 4:161–171, 1998.

M.G.C. Resende and T.A. Feo. A GRASP for satisfiability. In D.S. Johnson and M.A. Trick, editors, *Cliques, Coloring, and Satisfiability: The Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 499–520. American Mathematical Society, 1996.

M.G.C. Resende, T.A. Feo, and S.H. Smith. Algorithm 787: Fortran subroutines for approximate solution of maximum independent set problems using GRASP. *ACM Trans. Math. Software*, 24:386–394, 1998.

M.G.C. Resende, P.M. Pardalos, and Y. Li. Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software*, 22:104–118, 1996.

M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Approximate solution of weighted MAX-SAT problems using GRASP. In J. Gu and P.M. Pardalos, editors, *Satisfiability problems*, volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 393–405. American Mathematical Society, 1997.

M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Fortran subroutines for computing approximate solutions of MAX-SAT problems using GRASP. *Discrete Applied Mathematics*, 100:95–113, 2000.

M.G.C. Resende and C.C. Ribeiro. A GRASP for graph planarization. *Networks*, 29: 173–189, 1997.

C.C. Ribeiro and M.G.C. Resende. Algorithm 797: Fortran subroutines for approximate solution of graph planarization problems using GRASP. *ACM Transactions on Mathematical Software*, 25:341–352, 1999.

B. Selman, H.A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the AAAI-94*, pages 337–343. MIT Press, 1994.

E.D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.

M.G.A. Verhoeven and E.H.L. Aarts. Parallel local search. *J. of Heuristics*, 1:43–66, 1995.

Appendix: TTT plots

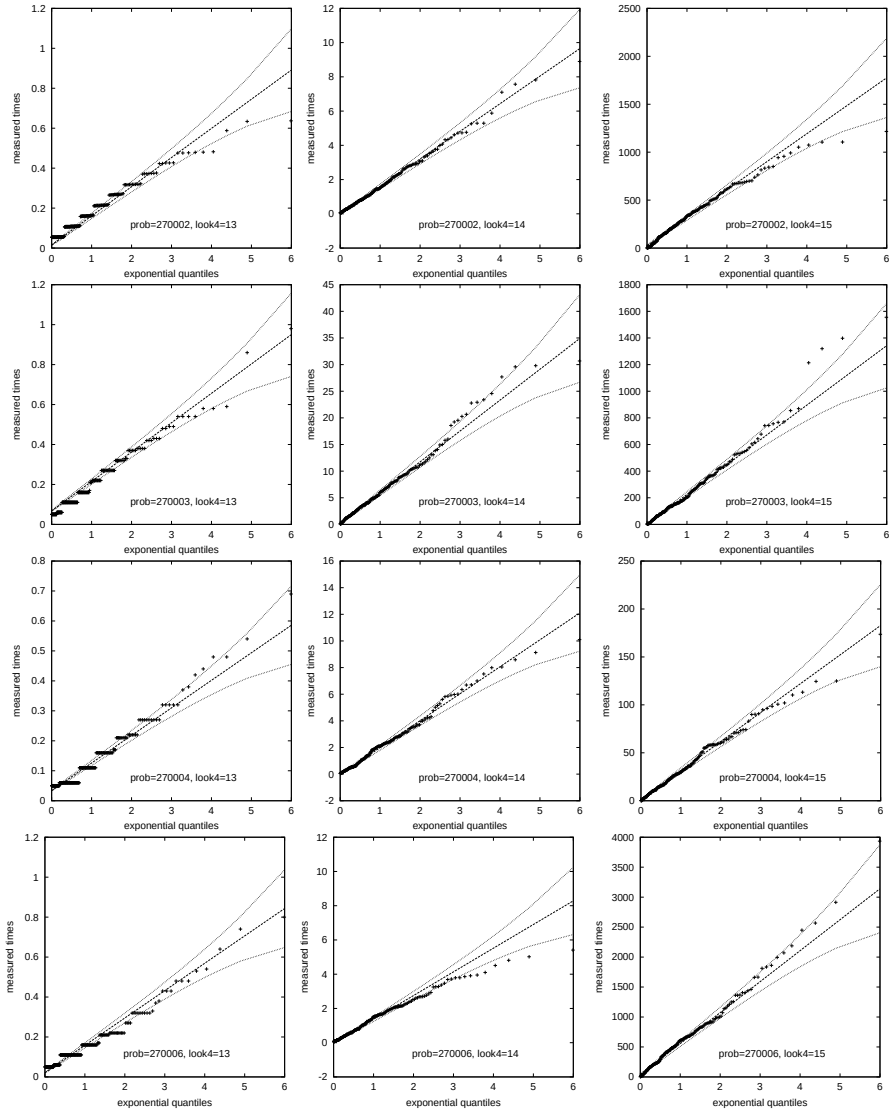


Figure A.1 Q-Q plots for GRASP for maximum independent set

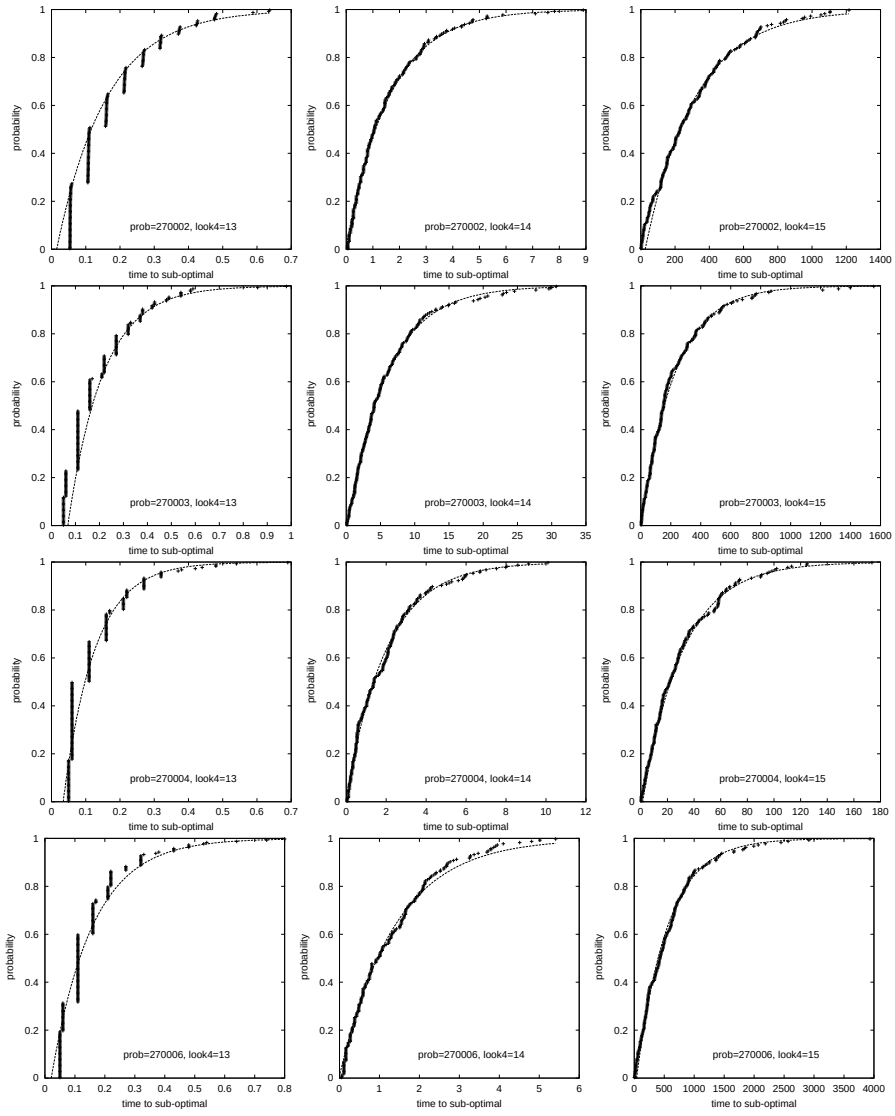


Figure A.2 Exponential plots for GRASP for maximum independent set

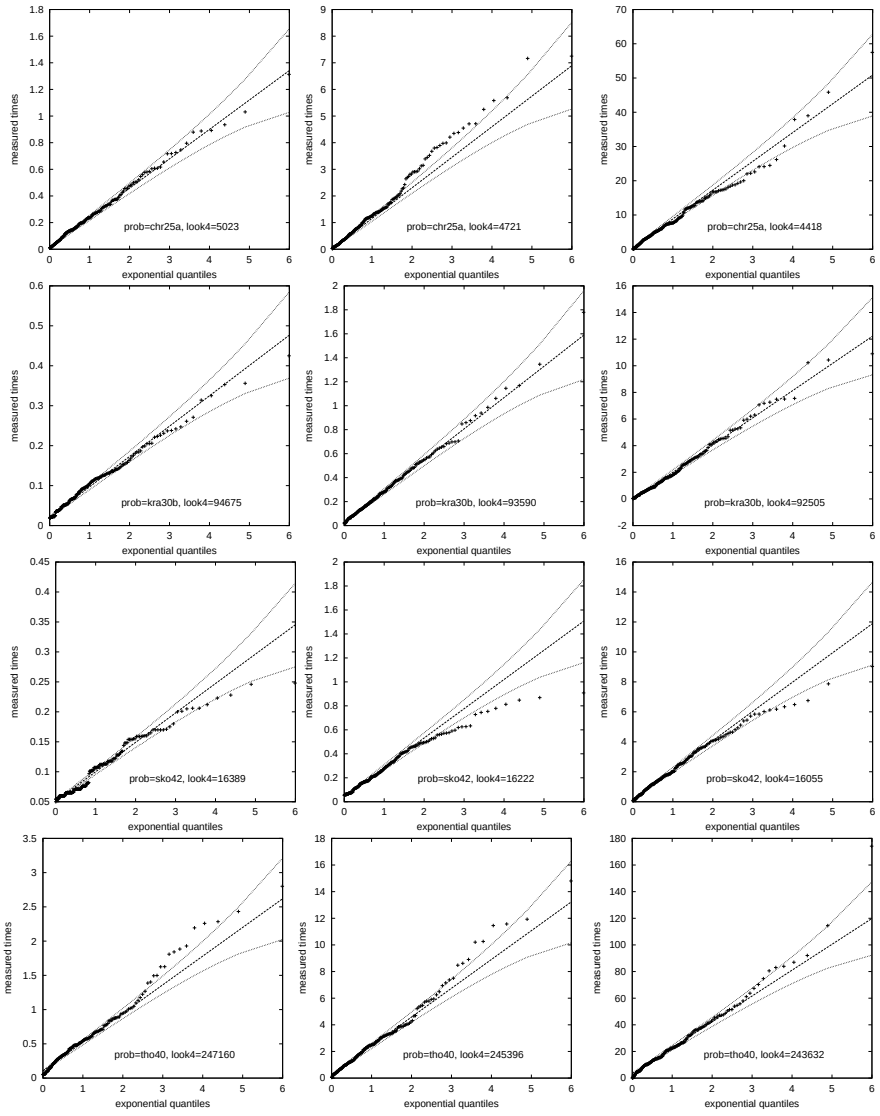


Figure A.3 Q-Q plots for GRASP for quadratic assignment

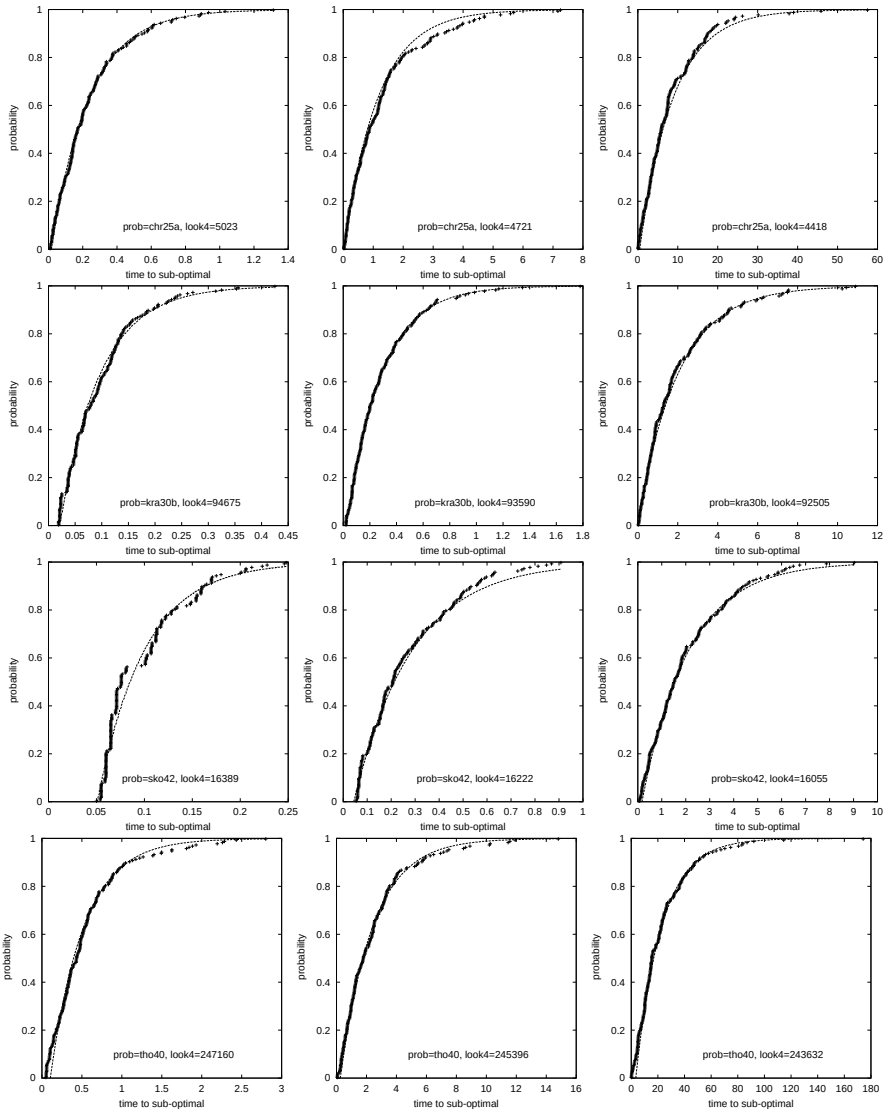


Figure A.4 Exponential plots for GRASP for quadratic assignment

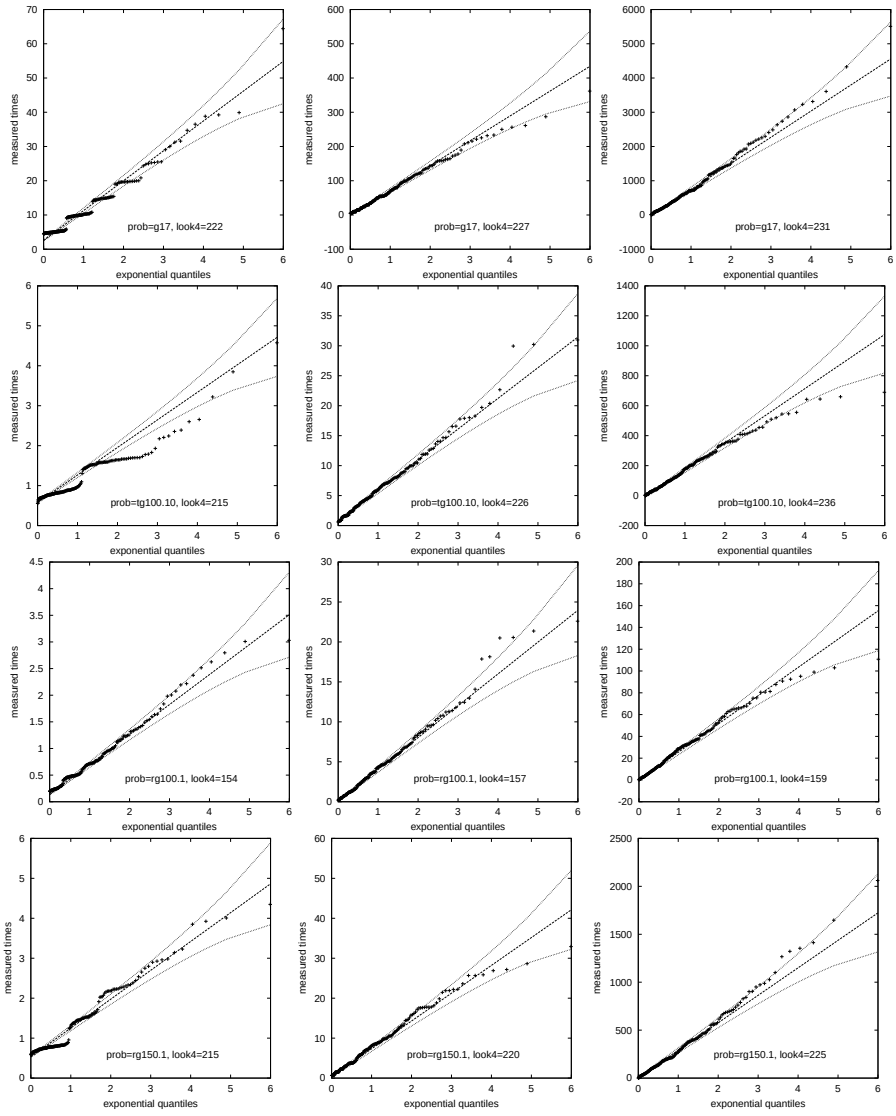


Figure A.5 Q-Q plots for GRASP for graph planarization

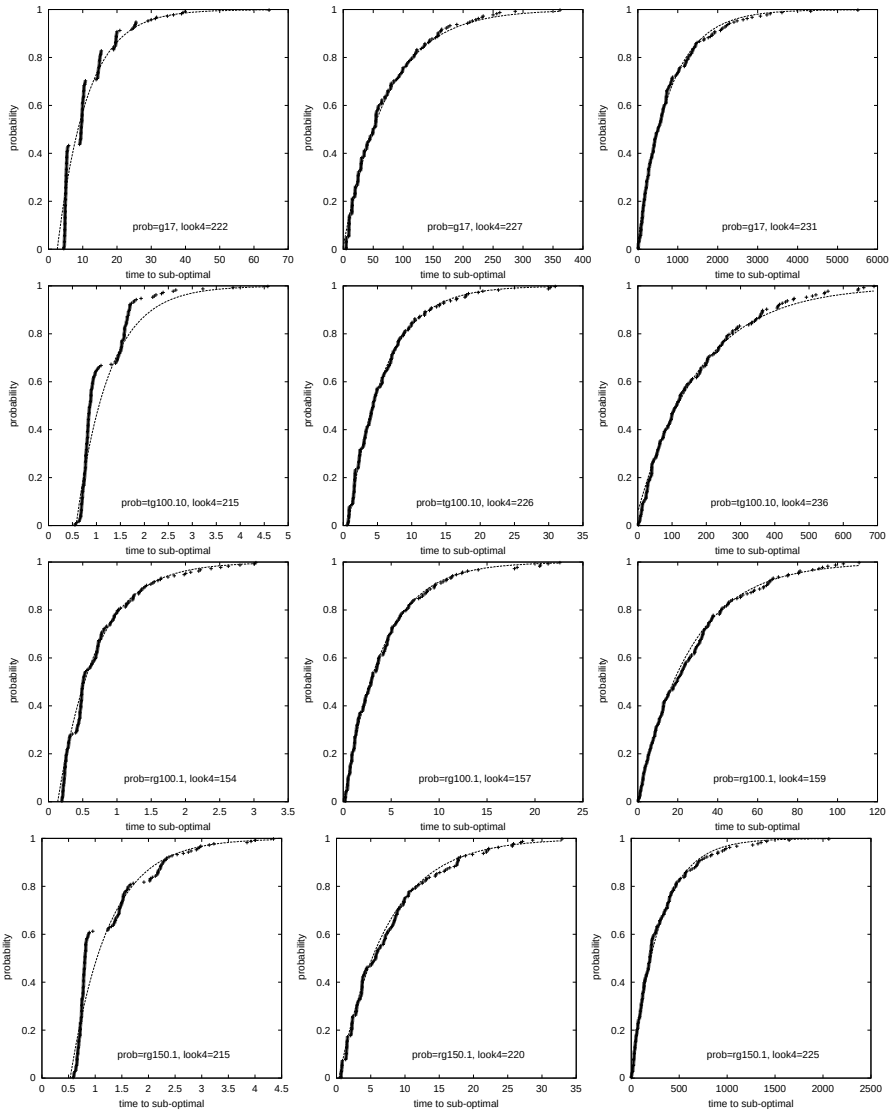


Figure A.6 Exponential plots for GRASP for graph planarization

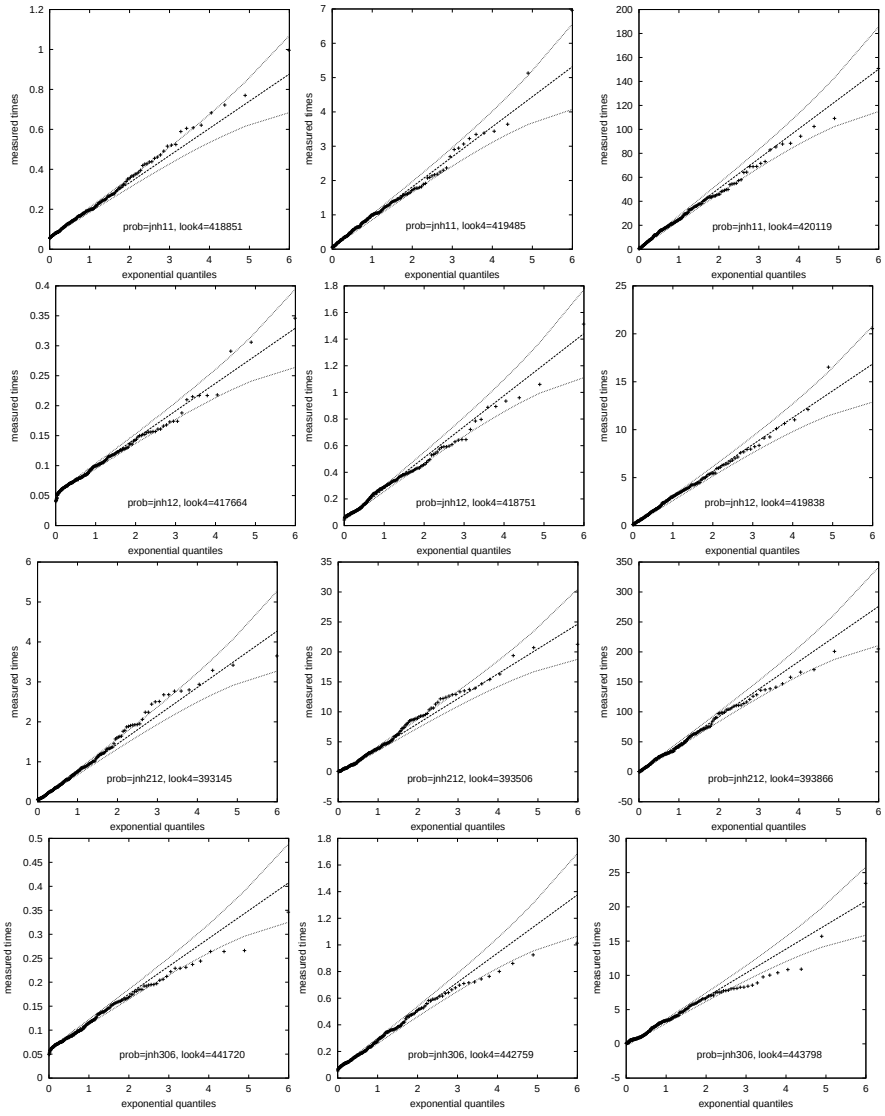


Figure A.7 Q-Q plots for GRASP for maximum weighted satisfiability

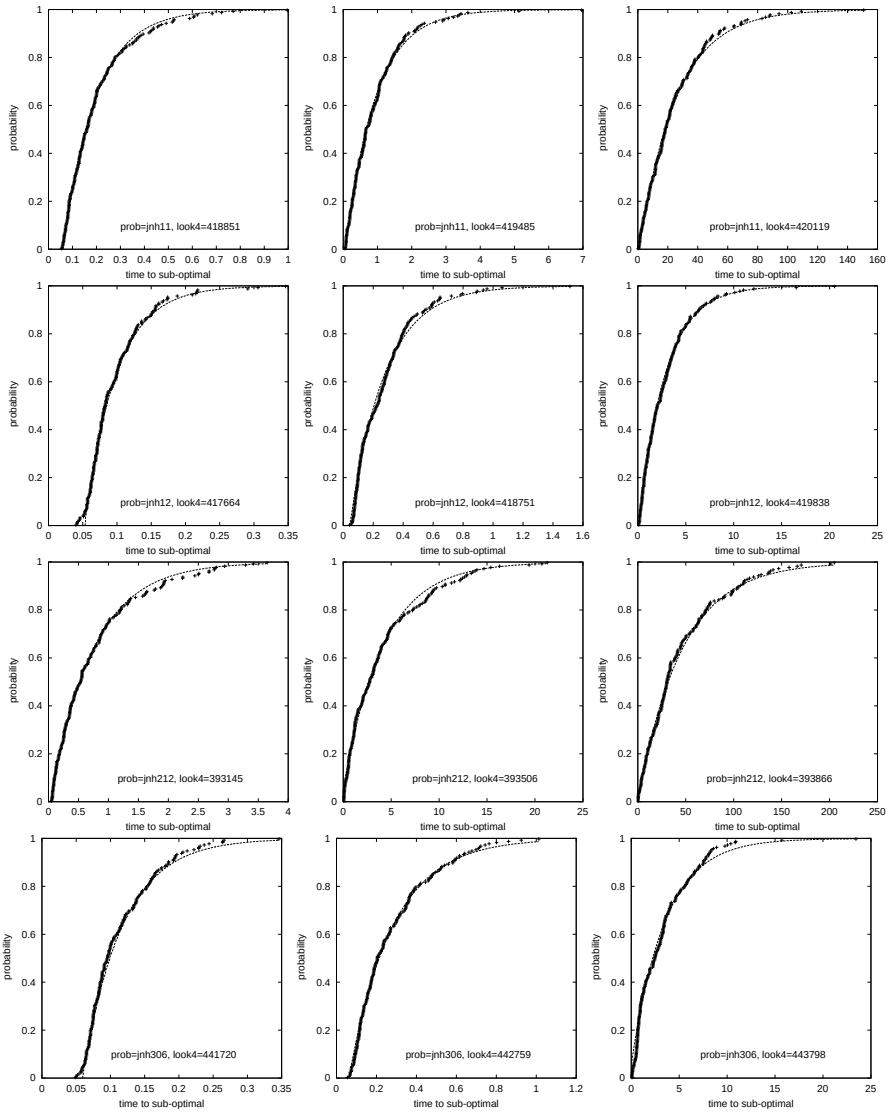


Figure A.8 Exponential plots for GRASP for maximum weighted satisfiability

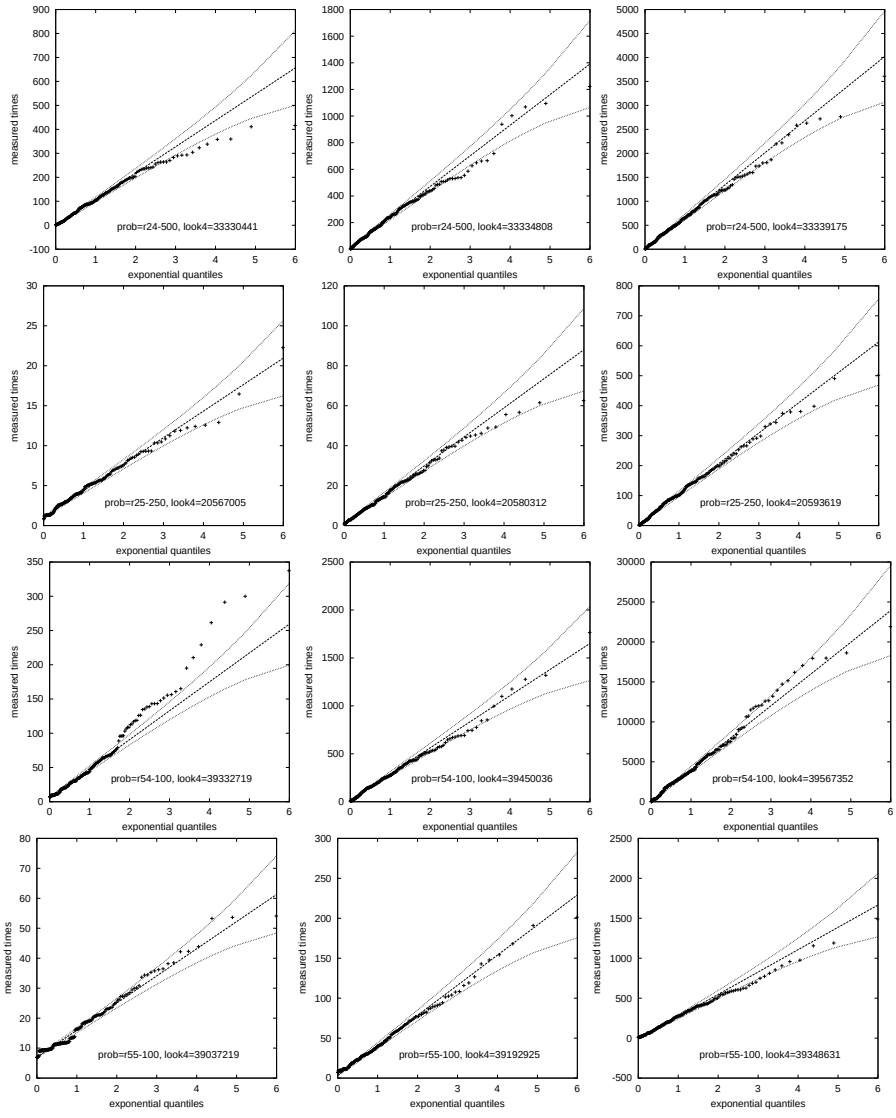


Figure A.9 Q-Q plots for GRASP for maximum covering

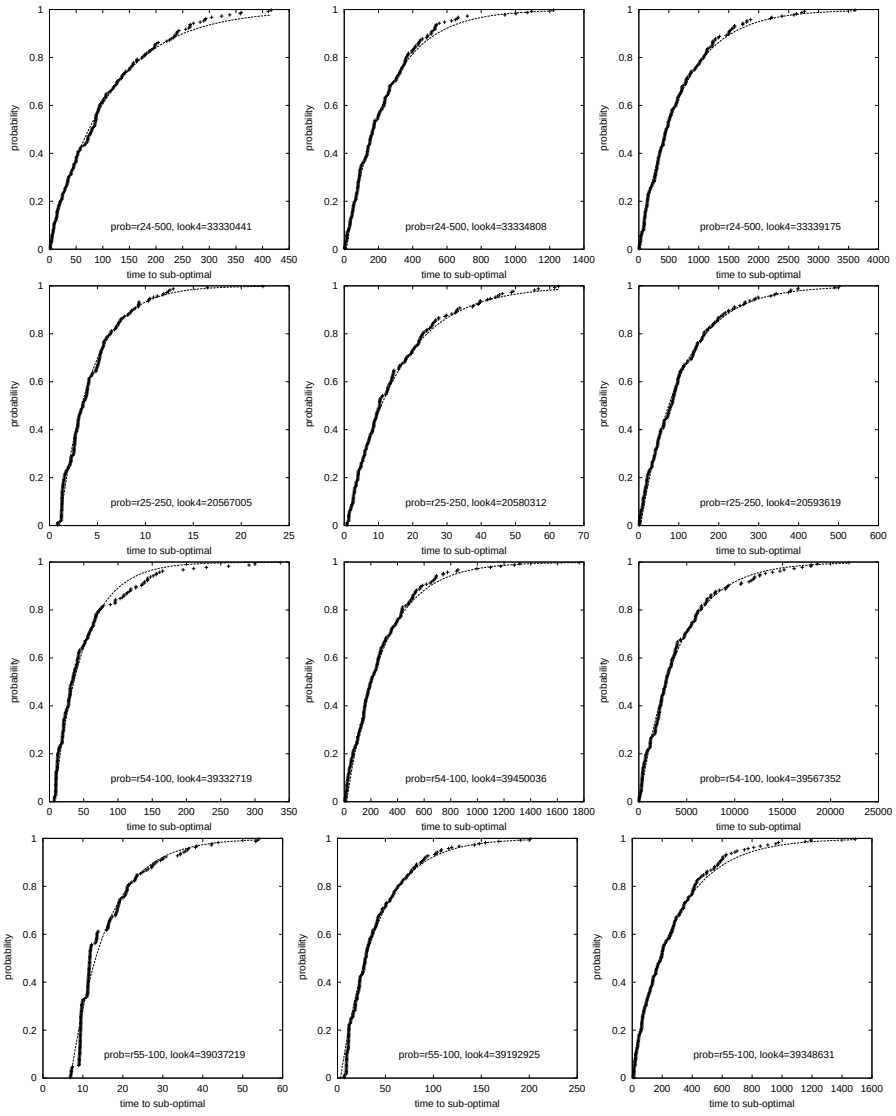


Figure A.10 Exponential plots for GRASP for maximum covering

7 PARALLEL GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURES

Mauricio G. C. Resende[1], and Celso C. Ribeiro[2]

[1]Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA.
mgcr@research.att.com

[2]Department of Computer Science
Universidade Federal Fluminense
Niterói, RJ 24210-240, Brazil
celso@ic.uff.br

Abstract: A GRASP (Greedy Randomized Adaptive Search Procedure) is a metaheuristic for producing good-quality solutions of combinatorial optimization problems. It is usually implemented with a construction procedure based on a greedy randomized algorithm followed by local search. In this Chapter, we survey parallel implementations of GRASP. We describe simple strategies to implement independent parallel GRASP heuristics and more complex cooperative schemes using a pool of elite solutions to intensify the search process. Some applications of independent and cooperative parallelizations are presented in detail.
Keywords: Combinatorial optimization, local search, GRASP, path-relinking, parallel algorithm.

7.1 INTRODUCTION

Metaheuristics are high level procedures that coordinate simple heuristics, such as local search, to find solutions that are of better quality than those found by the simple heuristics alone. One such metaheuristic is GRASP (Greedy Randomized Adaptive Search Procedure) (Feo and Resende, 1989; 1995; Festa and Resende, 2002; Resende and Ribeiro, 2002). A GRASP is a multi-start procedure, where each iteration usually consists of two phases: construction and local search. The construction phase

produces a feasible solution that is used as the starting point for local search. The multi-start procedure returns the best local optimum found.

In the GRASP construction phase, a feasible solution is built, one element at a time. For example, a spanning tree is built one edge at a time; a schedule is built one operation at a time; and a clique is built one vertex at a time. The set of *candidate elements* is made up of those elements that can be added to the current solution under construction without causing infeasibilities. When building a spanning tree, for example, the candidate elements are those yet unselected edges whose inclusion in the solution does not result in a cycle. A candidate element is evaluated by a *greedy function* that measures the local benefit of including that element in the partially constructed solution. The value-based *restricted candidate list* (RCL) is made up of candidate elements having a greedy function value at least as good as a specified threshold. The next element to be included in the solution is selected at random from the RCL. Its inclusion in the solution alters the greedy function and the set of candidate elements used to determine the next RCL. The construction procedure terminates when the set of candidate elements is empty, obtaining a feasible solution. Algorithm 4 shows a GRASP in pseudo-code form, where the objective function $f(x)$ is minimized over the set X. The GRASP runs for `MaxIterations` iterations. The best solution returned is x^*, with $f(x^*) = f^*$.

Data : Number of iterations `MaxIterations`
Result : Solution $x^* \in X$
$f^* \leftarrow \infty$;
for $i = 1, \ldots, i_{\max}$ **do**
  $x \leftarrow$ `GreedyRandomizedConstruction`$()$;
  $x \leftarrow$ `LocalSearch`(x);
  if $f(x) < f^*$ **then**
    $f^* \leftarrow f(x)$;
    $x^* \leftarrow x$;
  end
end

Algorithm 4: Pseudo-code of a basic GRASP for minimization.

Local search makes use of the concept of solution neighborhood. A local search algorithm successively replaces the current solution by a better solution in its neighborhood, if one exists. It terminates with a locally optimal solution when there is no better solution in the neighborhood. Since the solutions generated by a GRASP construction phase are usually sub-optimal, local search almost always improves the constructed solution.

GRASP has been used to find quality solutions for a wide range of combinatorial optimization problems (Festa and Resende, 2002; 2004). Many extensions and improvements with respect to the GRASP introduced in Feo and Resende (1989) and Feo and Resende (1995) have been proposed. Many of these extensions consist in the hybridization of the method with other methaheuristics.

Parallel computers have increasingly found their way into metaheuristics (Cung et al., 2002; Duni et al., 2002). Most of the parallel implementations of GRASP found in the literature consist in either partitioning the search space or partitioning

the GRASP iterations and assigning each partition to a processor (Alvim and Ribeiro, 1998; Alvim, 1998; Feo et al., 1994; Drummond et al., 2002; Li et al., 1994; Martins et al., 1999; 2000; 1998; Murphey et al., 1998; Pardalos et al., 1995; 1996; Resende et al., 1998). GRASP is applied to each partition in parallel. These implementations can be categorized as *multiple-walk independent-thread* (Cung et al., 2002; Verhoeven and Aarts, 1995), where the communication among processors during GRASP iterations is limited to the detection of program termination,

Recently, there has been much work on hydridization of GRASP and path-relinking (Resende and Ribeiro, 2005). Parallel approaches for GRASP with path-relinking can be categorized as multiple-walk independent-thread or *multiple-walk cooperative-thread* (Cung et al., 2002; Verhoeven and Aarts, 1995), where processors share information on elite solutions visited during previous GRASP iterations. Examples of parallel GRASP with path-relinking can be found in (Aiex et al., 2003; 2005; Canuto et al., 2001; Martins et al., 2004; Ribeiro and Rosseti, 2002).

In this Chapter, we present a survey of parallel GRASP heuristics. In Section 7.2, we consider multiple-walk independent-thread strategies. Multiple-walk cooperative-thread strategies are examined in Section 7.3. Some applications of parallel GRASP and parallel GRASP with path-relinking are surveyed in Section 7.4. In Section 7.5, we make some concluding remarks.

7.2 MULTIPLE-WALK INDEPENDENT-THREAD STRATEGIES

Most parallel implementations of GRASP follow the *multiple-walk independent-thread* strategy, based on the distribution of the iterations over the processors. In general, each search thread has to perform `MaxIterations`$/p$ iterations, where p and `MaxIterations` are, respectively, the number of processors and the total number of iterations. Each processor has a copy of the sequential algorithm, a copy of the problem data, and an independent seed to generate its own pseudorandom number sequence. To avoid that the processors find the same solutions, each of them must use a different sequence of pseudorandom numbers. A single global variable is required to store the best solution found over all processors. One of the processors acts as the master, reading and distributing problem data, generating the seeds which will be used by the pseudorandom number generators at each processor, distributing the iterations, and collecting the best solution found by each processor. Since the iterations are completely independent and very little information is exchanged, linear speedups are easily obtained provided that no major load imbalance problems occur. The iterations may be evenly distributed over the processors or according with their demands, to improve load balancing.

Pardalos et al. (1995) reported on results of a parallel GRASP for the quadratic assignment problem on a Kendall Square Research KSR-1 parallel computer with 128 processors. The implementation used the `pthread` mechanism, a lightweight process that is the fundamental unit of concurrency on the KSR-1 (Kendall Square Research, 1992). Each pthread executes on a separate processor and has its own memory. Twenty instances from the QAPLIB (Burkard et al., 1991) were run for 1000 GRASP iterations on each of 64 single processors. For each instance, the best solution found over all processors was used as the stopping criterion for solving the instance on 54, 44, 34,

24, 14, 4, and 1 processors. Speedups were computed by averaging the running times of all instances.

Pardalos et al. (1996) implemented a parallel GRASP for the MAX-SAT problem on a cluster of SUN-SPARC 10 workstations, sharing the same file system, with communication done using the Parallel Virtual Machine (PVM) (Geist et al., 1994) software package. Each instance was run on a parallel GRASP using 1, 5, 10, and 15 processors, with a maximum number of iterations of 1000, 200, 100, and 66, respectively. The amount of CPU time required to perform the specified number of iterations, and the best solution found were recorded. Since communication was kept to a minimum, linear speedups were expected. Figure 7.1 shows individual speedups as well as average speedups for these runs. Figure 7.2 shows that the average quality of the solution found was not greatly affected by the number of processors used.

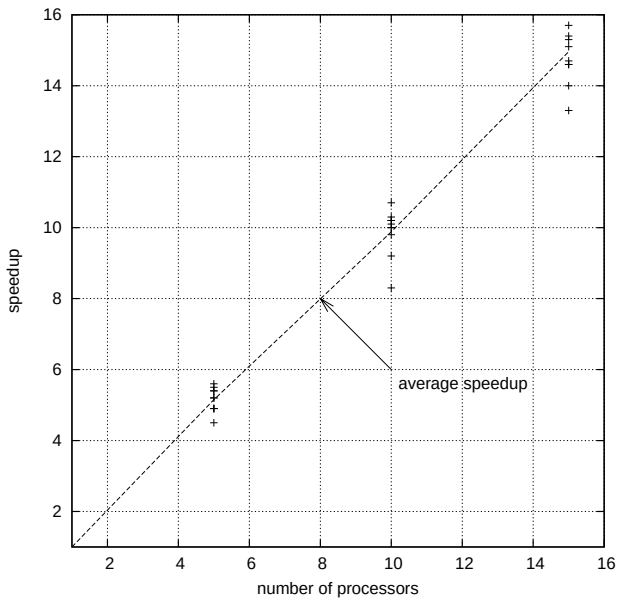


Figure 7.1 Average speedups on 5, 10, and 15 processors for maximum satisfiability problems.

Martins et al. (1998) implemented a parallel GRASP for the Steiner problem in graphs. Parallelization is achieved by the distribution of 512 iterations over the processors, with the value of the RCL parameter α randomly chosen in the interval $[0.0, 0.3]$ at each iteration. The algorithm was tested on an IBM SP-2 machine with 32 processors, using the Message Passing Interface (MPI) library (Snir et al., 1998) for communication. The 60 problems from series C, D, and E of the OR-Library (Beasley, 1990) were used for the computational experiments. The parallel implementation obtained 45 optimal solutions over the 60 test instances. The relative deviation with respect to the optimal value was never larger than 4%. Almost-linear speedups observed for 2,

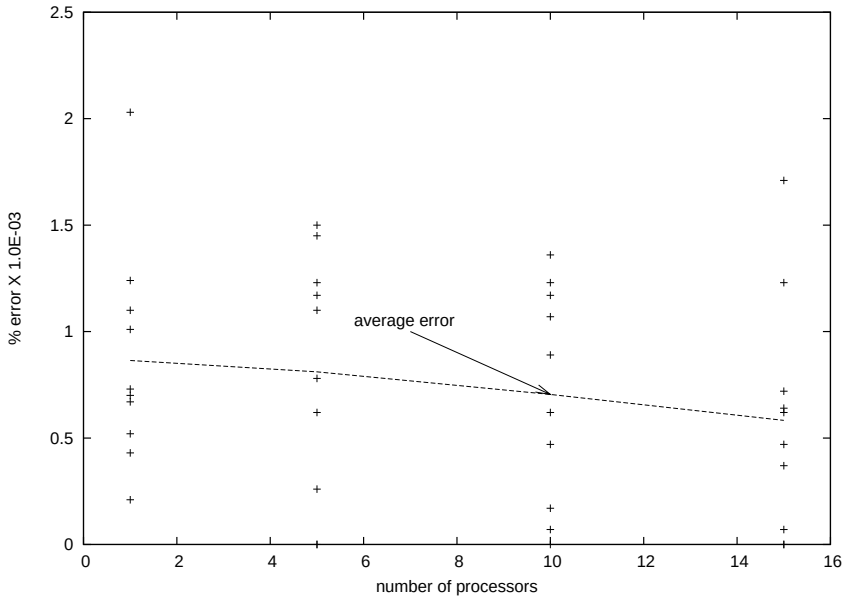


Figure 7.2 Percentage error on 1, 5, 10, and 15 processors for maximum satisfiability problems.

4, 8, and 16 processors with respect to the sequential implementation are illustrated in Figure 7.3.

Path-relinking may also be used in conjunction with parallel implementations of GRASP. In the case of the multiple-walk independent-thread implementation described by Aiex et al. (2005) for the 3-index assignment problem and Aiex et al. (2003) for the job shop scheduling problem, each processor applies path-relinking to pairs of elite solutions stored in a local pool. Computational results using MPI on an SGI Challenge computer with 28 R10000 processors showed linear speedups for the 3-index assignment problem, but sub-linear for the job shop scheduling problem.

Alvim and Ribeiro (1998); Alvim (1998) showed that multiple-walk independent-thread approaches for the parallelization of GRASP may benefit much from load balancing techniques, whenever heterogeneous processors are used or if the parallel machine is simultaneously shared by several users. In this case, almost-linear speedups may be obtained with a heterogeneous distribution of the iterations over the p processors in $q \geq p$ packets. Each processor starts performing one packet of $\lceil \texttt{MaxIterations}/q \rceil$ iterations and informs the master when it finishes its packet of iterations. The master stops the execution of each slave processor when there are no more iterations to be performed and collects the best solution found. Faster or less loaded processors will perform more iterations than the others. In the case of the parallel GRASP implemented for the problem of traffic assignment described in Prais and Ribeiro (2000), this dynamic load balancing strategy allowed reductions in the elapsed

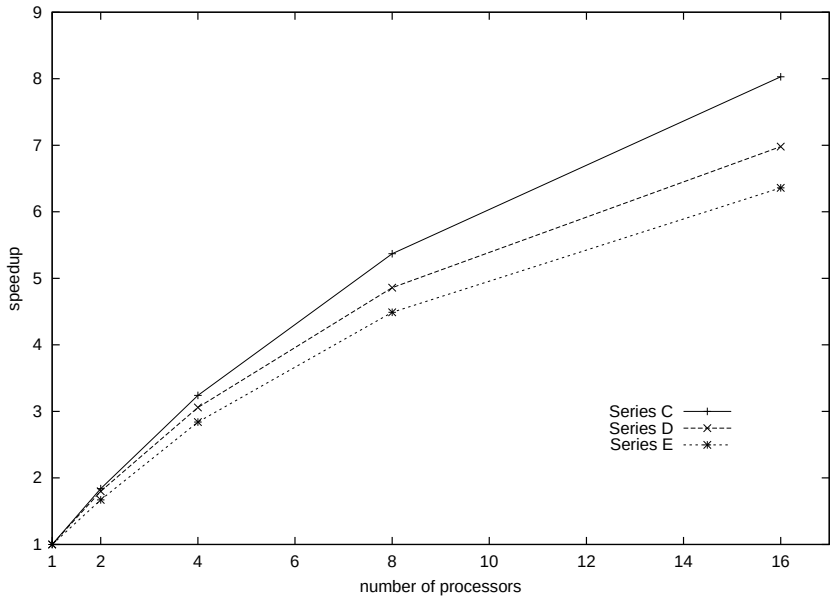


Figure 7.3 Average speedups on 2, 4, 8, and 16 processors on Steiner tree problem in graphs.

times of up to 15% with respect to the times observed for the static strategy, in which the iterations were uniformly distributed over the processors.

The efficiency of multiple-walk independent-thread parallel implementations of metaheuristics, based on running multiple copies of the same sequential algorithm, has been addressed by some authors. A given target value τ for the objective function is broadcast to all processors which independently execute the sequential algorithm. All processors halt immediately after one of them finds a solution with value at least as good as τ. The speedup is given by the ratio between the times needed to find a solution with value at least as good as τ, using respectively the sequential algorithm and the parallel implementation with p processors. These speedups are linear for a number of metaheuristics, including simulated annealing (Dodd, 1990; Osborne and Gillett, 1991); iterated local search algorithms for the traveling salesman problem (Eikelder et al., 1996); tabu search, provided that the search starts from a local optimum (Battiti and Tecchiolli, 1992; Taillard, 1991); and WalkSAT (Selman et al., 1994) on hard random 3-SAT problems (Hoos and Stützle, 1999). This observation can be explained if the random variable *time to find a solution within some target value* is exponentially distributed, as indicated by the following proposition (Verhoeven and Aarts, 1995):
Proposition 1: Let $P_\rho(t)$ be the probability of not having found a given target solution value in t time units with ρ independent processes. If $P_1(t) = e^{-t/\lambda}$ with $\lambda \in \mathbb{R}^+$, corresponding to an exponential distribution, then $P_\rho(t) = e^{-\rho t/\lambda}$.

This proposition follows from the definition of the exponential distribution. It implies that the probability $1 - e^{-\rho t/\lambda}$ of finding a solution within a given target value

in time ρt with a sequential algorithm is equal to the probability of finding a solution at least as good as that in time t using ρ independent parallel processors. Hence, it is possible to achieve linear speedups in the time to find a solution within a target value by multiple independent processors. An analogous proposition can be stated for a two parameter (shifted) exponential distribution:

Proposition 2: Let $P_\rho(t)$ be the probability of not having found a given target solution value in t time units with ρ independent processors. If $P_1(t) = e^{-(t-\mu)/\lambda}$ with $\lambda \in \mathbb{R}^+$ and $\mu \in \mathbb{R}^+$, corresponding to a two parameter exponential distribution, then $P_\rho(t) = e^{-\rho(t-\mu)/\lambda}$.

Analogously, this proposition follows from the definition of the two-parameter exponential distribution. It implies that the probability of finding a solution within a given target value in time ρt with a sequential algorithm is equal to $1 - e^{-(\rho t-\mu)/\lambda}$, while the probability of finding a solution at least as good as that in time t using ρ independent parallel processors is $1 - e^{-\rho(t-\mu)/\lambda}$. If $\mu = 0$, then both probabilities are equal and correspond to the non-shifted exponential distribution. Furthermore, if $\rho\mu \ll \lambda$, then the two probabilities are approximately equal and it is possible to approximately achieve linear speedups in the time to find a solution within a target value using multiple independent processors.

Aiex et al. (2002) showed experimentally that the solution times for GRASP also have this property, i.e. that they fit a two-parameter exponential distribution. Figure 7.4 illustrates this result, depicting the superimposed empirical and theoretical distributions observed for one of the cases studied along the computational experiments reported by the authors, which involved 2400 runs of GRASP procedures for each of five different problems: maximum independent set (Feo et al., 1994; Resende et al., 1998), quadratic assignment (Li et al., 1994; Resende et al., 1996), graph planarization (Resende and Ribeiro, 1997; Ribeiro and Resende, 1999), maximum weighted satisfiability (Resende et al., 2000), and maximum covering (Resende, 1998). We observe that the empirical distribution plots illustrating these conclusions were originally introduced by Feo et al. (1994). Empirical distributions are produced from experimental data and corresponding theoretical distributions are estimated from the empirical distributions. The same result still holds when GRASP is implemented in conjunction with a post-optimization path-relinking procedure (Aiex et al., 2005).

A quantile-quantile plot (Q-Q plot) and a plot showing the empirical and the theoretical distributions of the random variable time to target value for the sequential GRASP and GRASP with path-relinking for the three-index assignment problem (Aiex et al., 2005) are shown in Figures 7.5 and 7.6, respectively. Analogously, Figures 7.7 and 7.8 show the same plots for the job-shop scheduling problem (Aiex et al., 2003). These plots are computed by running the algorithms for 200 independent runs. Each run ends when the algorithm finds a solution with value less than or equal to a specified target value. Each running time is recorded and the times are sorted in increasing order. We associate with the i-th sorted running time (t_i) a probability $p_i = (i - \frac{1}{2})/200$, and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 200$ as the empirical distribution.

Following Chambers et al. (1983), one determines the theoretical quantile-quantile plot for the data to estimate the parameters of the two-parameter exponential distribution. To describe Q-Q plots, recall that the cumulative distribution function for the

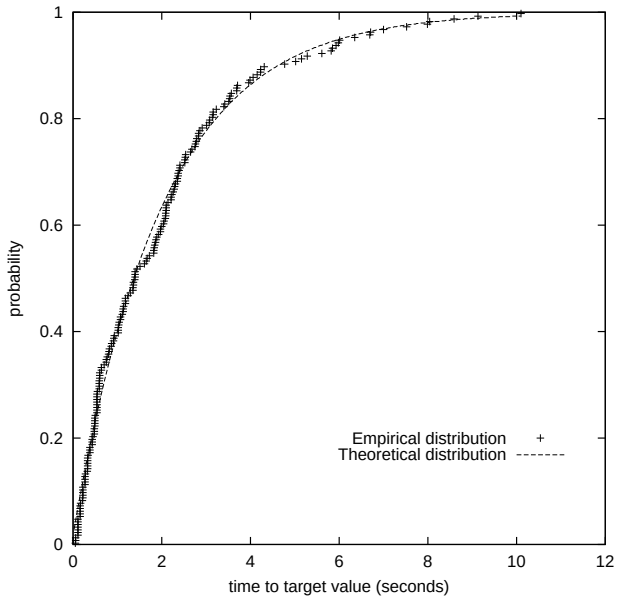


Figure 7.4 Superimposed empirical and theoretical distributions (times to target values measured in seconds on an SGI Challenge computer with 28 processors).

two-parameter exponential distribution is given by

$$F(t) = 1 - e^{-(t-\mu)/\lambda},$$

where λ is the mean and standard deviation of the distribution data and μ is the shift of the distribution with respect to the ordinate axis. For each value p_i, $i = 1, \ldots, 200$, we associate a p_i-quantile $Qt(p_i)$ of the theoretical distribution. For each p_i-quantile we have, by definition, that

$$F((Qt(p_i)) = p_i.$$

Hence, $Qt(p_i) = F^{-1}(p_i)$ and therefore, for the two-parameter exponential distribution, we have

$$Qt(p_i) = -\lambda \ln(1 - p_i) + \mu.$$

The quantiles of the data of an empirical distribution are simply the (sorted) raw data.

A theoretical quantile-quantile plot (or theoretical Q-Q plot) is obtained by plotting the quantiles of the data of an empirical distribution against the quantiles of a theoretical distribution. This involves three steps. First, the data (in this case, the measured times) are sorted in ascending order. Second, the quantiles of the theoretical exponential distribution are obtained. Finally, a plot of the data against the theoretical quantiles is made.

When the theoretical distribution is a close approximation of the empirical distribution, the points in the Q-Q plot will have a nearly straight configuration. If the parameters λ and μ of the theoretical distribution that best fits the measured data could

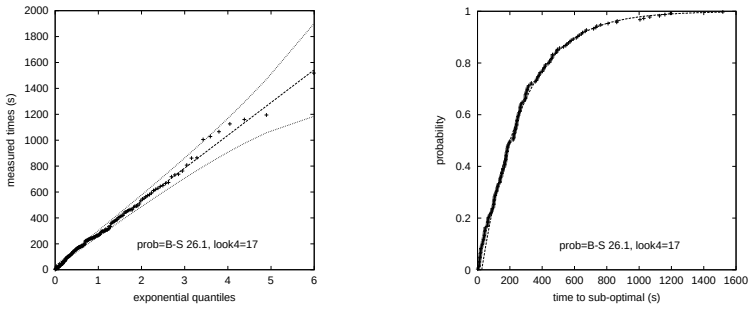


Figure 7.5 Q-Q plot and exponential distribution for GRASP for the three-index assignment problem: instance `B-S 26.1` with target value of 17.

be estimated a priori, the points in a Q-Q plot would tend to follow the line $x = y$. Alternatively, in a plot of the data against a two-parameter exponential distribution with $\lambda' = 1$ and $\mu' = 0$, the points would tend to follow the line $y = \lambda x + \mu$. Consequently, parameters λ and μ of the two-parameter exponential distribution can be estimated, respectively, by the slope and the intercept of the line depicted in the Q-Q plot.

To avoid possible distortions caused by outliers, one does not estimate the distribution mean by linear regression on the points of the Q-Q plot. Instead, one estimates the slope $\hat{\lambda}$ of line $y = \lambda x + \mu$ using the upper quartile q_u and lower quartile q_l of the data. The upper and lower quartiles are, respectively, the $Q(\frac{1}{4})$ and $Q(\frac{3}{4})$ quantiles, respectively. Let

$$\hat{\lambda} = (z_u - z_l)/(q_u - q_l)$$

be an estimate of the slope, where z_u and z_l are the u-th and l-th points of the ordered measured times, respectively. These estimates are used to plot the theoretical distributions on the plots on the right side of the figures.

The lines above and below the estimated line on the Q-Q plots correspond to plus and minus one standard deviation in the vertical direction from the line fitted to the plot. This superimposed variability information is used to analyze the straightness of the Q-Q plots.

Aiex and Resende (2005) proposed a test using a sequential implementation to determine whether it is likely that a parallel implementation using multiple independent processors will be efficient. A parallel implementation is said to be efficient if it achieves linear speedup (with respect to wall time) to find a solution at least as good as a given target value. The test consists in running K (200, for example) independent trials of the sequential program to build a Q-Q plot and estimate the parameters μ and λ of the shifted exponential distribution. If $\rho|\mu| \ll \lambda$, then we predict that the parallel implementation will be efficient.

7.3 MULTIPLE-WALK COOPERATIVE-THREAD STRATEGIES

Path-relinking has been implemented with GRASP in multiple-walk independent-thread strategies (Aiex et al., 2005). In this section, however, we focus on the use

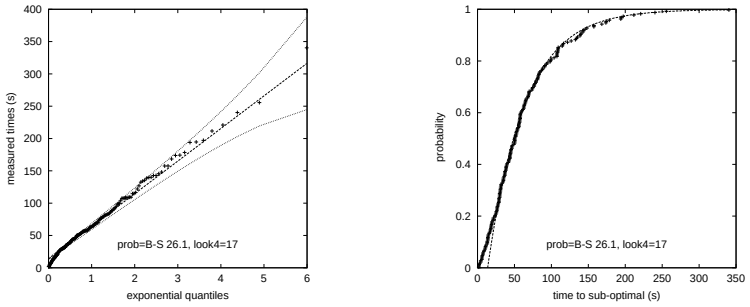


Figure 7.6 Q-Q plot and exponential distribution for GRASP with path-relinking for the three-index assignment problem: instance `B-S 26.1` with target value of 17.

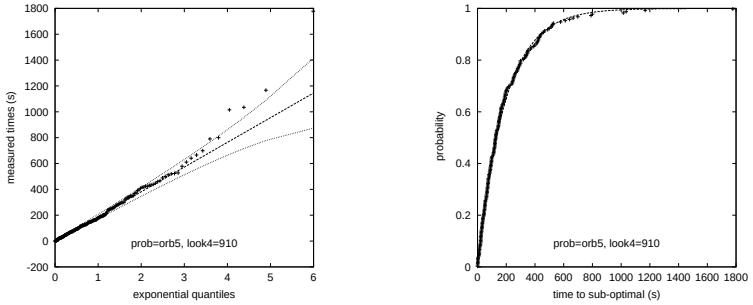


Figure 7.7 Q-Q plot and exponential distribution for GRASP for the job shop scheduling problem: instance `orb5` with target value of 910.

of path-relinking as a mechanism for implementing GRASP in the multiple-walk cooperative-thread strategies framework. We first briefly outline path-relinking and its hybridization with GRASP. Then, we discuss how cooperation among the threads can be achieved by using path-relinking.

Path-relinking was originally proposed by Glover (1996) as a strategy to explore trajectories connecting elite solutions obtained by tabu search or scatter search (Glover, 2000; Glover and Laguna, 1997; Glover et al., 2000). Paths in the solution space connecting pairs of elite solutions are explored in the search for better solutions. Each pair consists of a starting solution and a guiding solution. Paths emanating from the starting solution are generated by applying moves that introduce in the current solution attributes that are present in the guiding solution.

Algorithm 5 shows the pseudo-code of the path-relinking procedure applied between the starting and guiding solutions. The procedure first computes the symmetric difference $\Delta(x_s, x_t)$ between the two solutions, which defines the moves needed to reach the guiding solution (x_t) from the initial solution (x_s). A path of neighboring

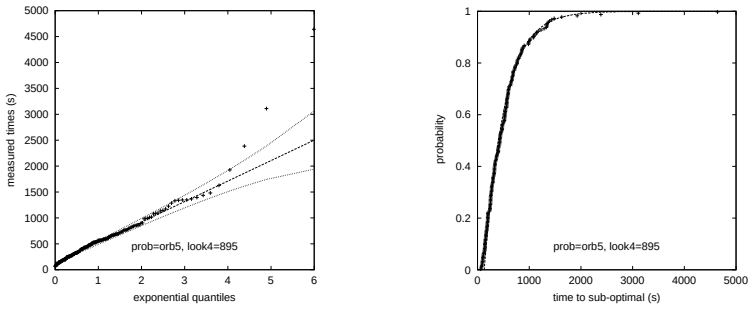


Figure 7.8 Q-Q plot and exponential distribution for GRASP with path-relinking for the job shop scheduling problem: instance `orb5` with target value of 895.

Data : Starting solution x_s and guiding solution x_t
Result : Best solution x^* in path from x_s to x_t
Compute symmetric difference $\Delta(x_s, x_t)$;
$f^* \leftarrow \min\{f(x_s), f(x_t)\}$;
$x^* \leftarrow \text{argmin}\{f(x_s), f(x_t)\}$;
$x \leftarrow x_s$;
while $\Delta(x, x_t) \neq \emptyset$ **do**
  $m^* \leftarrow \text{argmin}\{f(x \oplus m) : m \in \Delta(x, x_t)\}$;
  $\Delta(x \oplus m^*, x_t) \leftarrow \Delta(x, x_t) \setminus \{m^*\}$;
  $x \leftarrow x \oplus m^*$;
  if $f(x) < f^*$ **then**
    $f^* \leftarrow f(x)$;
    $x^* \leftarrow x$;
  end
end

Algorithm 5: Pseudo-code of path-relinking from starting solution x_s to guiding solution x_t.

solutions is generated linking x_s and x_t. The best solution x^* in this path is returned. At each step, all moves $m \in \Delta(x, x_t)$ from the current solution x are examined and the one which results in the least cost solution is selected, i.e. the move that minimizes $f(x \oplus m)$, where $x \oplus m$ is the solution resulting from applying move m to solution x. The best move m^* is made, producing solution $x \oplus m^*$. This move is taken out of the set of available moves. If necessary, the best solution x^* is updated. The procedure terminates when x_t is reached, i.e. when $\Delta(x, x_t) = \emptyset$.

The use of path-relinking within a GRASP procedure was first proposed by Laguna and Martí (1999). It was followed by several extensions, improvements, and successful applications (Aiex, 2002; Aiex et al., 2003; Aiex and Resende, 2005; Aiex et al., 2005; Binato et al., 2001; Canuto et al., 2001; Faria Jr. et al., 2005; Resende and

Ribeiro, 2003; 2005; Resende and Werneck, 2002; Ribeiro and Rosseti, 2002; Ribeiro et al., 2002; Rosseti, 2003).

In its hybridization with GRASP, path-relinking is usually applied to pairs (x,y) of solutions, where x is a locally optimal solution produced by each GRASP iteration after local search and y is an elite solution randomly chosen from a pool with a limited number `MaxElite` of elite solutions found along the search. Since the symmetric difference is a measure of the length of the path explored during relinking, a strategy biased toward pool elements y with high symmetric difference with respect to x is often better than one using uniform random selection (Resende and Werneck, 2002).

The pool is originally empty. To maintain a pool of good but diverse solutions, each locally optimal solution obtained by local search is considered as a candidate to be inserted into the pool if it is sufficiently different from every solution in the pool. If the pool already has `MaxElite` solutions and the candidate is better than the worst of them, then a simple strategy is to have the former replace the latter. Another strategy, which tends to increase the diversity of the pool, is to replace the pool element most similar to the candidate among all pool elements with cost worse than the candidate's. If the pool is not full, the candidate is simply inserted.

```
Data   : Number of iterations MaxIterations
Result : Solution x* ∈ X
P ← ∅;
f* ← ∞;
for i = 1,...,i_max do
    x ← GreedyRandomizedConstruction();
    x ← LocalSearch(x);
    if i ≥ 2 then
        Randomly select an elite subset Y ⊆ P to relink with x;
        for y ∈ Y do
            Set one of solutions x and y as the starting solution;
            Set the other as the guiding solution;
            x_p ← PathRelinking(x_s, x_t);
            Update the elite set P with x_p;
            if f(x_p) < f* then
                f* ← f(x_p);
                x* ← x_p;
            end
        end
    end
end
x* = argmin{f(x), x ∈ P};
```

Algorithm 6: A basic GRASP with path-relinking heuristic for minimization.

Algorithm 6 shows the pseudo-code for a hybrid GRASP with path-relinking. Each GRASP iteration has now three main steps. In the construction phase, a greedy randomized construction procedure is used to build a feasible solution. The local search phase takes the solution built in the first phase and progressively improves it using a neighborhood search strategy, until a local minimum is found. In the path-relinking phase, path-relinking is applied to the solution obtained by local search and to a randomly selected solution from the pool. The best solution found along this trajectory is also considered as a candidate for insertion in the pool and the incumbent is updated.

Two basic mechanisms may be used to implement a multiple-walk cooperative-thread GRASP with path-relinking heuristic. In *distributed strategies* (Aiex et al., 2003; Aiex and Resende, 2005), each thread maintains its own pool of elite solutions. Each iteration of each thread consists initially of a GRASP construction, followed by local search. Then, the local optimum is combined with a randomly selected element of the thread's pool using path-relinking. The output of path-relinking is finally tested for insertion into the pool. If accepted for insertion, the solution is sent to the other threads, where it is tested for insertion into the other pools. Collaboration takes place at this point. Though there may be some communication overhead in the early iterations, this tends to ease up as pool insertions become less frequent.

The second mechanism is that used in *centralized strategies* (Martins et al., 2004; Ribeiro and Rosseti, 2002), in which a single pool of elite solution is used. As before, each GRASP iteration performed at each thread starts by the construction and local search phases. Next, an elite solution is requested to and received from the centralized pool. Once path-relinking has been performed, the solution obtained as the output is sent to the pool and tested for insertion. Collaboration takes place when elite solutions are sent from the pool to other processors different from the one that originally computed it.

We notice that, in both the distributed and the centralized strategies, each processor has a copy of the sequential algorithm and a copy of the data. One processor acts as the master, reading and distributing the problem data, generating the seeds which will be used by the pseudo-random number generators at each processor, distributing the iterations, and collecting the best solution found by each processor. In the case of a distributed strategy, each processor has its own pool of elite solutions and all available processors perform GRASP iterations. Contrary to the case of a centralized strategy, one particular processor does not perform GRASP iterations and is used exclusively to store the pool and to handle all operations involving communication requests between the pool and the slaves. In the next section, we describe three examples of parallel implementations of GRASP with path-relinking.

7.4 SOME PARALLEL GRASP IMPLEMENTATIONS

In this section, we describe a comparison of multiple-walk independent-thread and multiple-walk cooperative-thread strategies for GRASP with path-relinking for the three-index assignment problem (Aiex et al., 2005), the job shop scheduling problem (Aiex et al., 2003), and the 2-path network design problem (Martins et al., 2004;

Ribeiro and Rosseti, 2002). For each problem, we first state the problem and describe the construction, local search, and path-relinking procedures. We then show numerical results comparing the different parallel implementations.

The experiments described in Subsections 7.4.1 and 7.4.2 were done on an SGI Challenge computer (16 196-MHz MIPS R10000 processors and 12 194-MHz MIPS R10000 processors) with 7.6 Gb of memory. The algorithms were coded in Fortran and were compiled with the SGI MIPSpro F77 compiler using flags `-O3 -static -u`. The parallel codes used SGI's Message Passing Toolkit 1.4, which contains a fully compliant implementation of version 1.2 of the Message-Passing Interface (MPI) (Snir et al., 1998) specification. In the parallel experiments, wall clock times were measured with the MPI function `MPI_WT`. This was also the case for runs with a single processor that are compared to multiple-processor runs. Timing in the parallel runs excludes the time to read the problem data, to initialize the random number generator seeds, and to output the solution.

In the experiments described in Subsection 7.4.3, both variants of the parallel GRASP with path-relining heuristic were implemented in C (version `egcs-2.91.66` of the `gcc` compiler) and the MPI LAM 6.3.2 implementation. Computational experiments were performed on a cluster of 32 Pentium II 400MHz processors with 32 Mbytes of RAM memory each, running under the Red Hat 6.2 implementation of Linux. Processors are connected by a 10 Mbits/s IBM 8274 switch.

7.4.1 Three-index assignment

7.4.1.1 Problem formulation. The NP-hard (Frieze, 1983; Garey and Johnson, 1979) three-index assignment problem (AP3) (Pierskalla, 1967) is a straightforward extension of the classical two-dimensional assignment problem and can be formulated as follows. Given three disjoint sets I, J, and K with $|I| = |J| = |K| = n$ and a weight c_{ijk} associated with each ordered triplet $(i, j, k) \in I \times J \times K$, find a minimum weight collection of n disjoint triplets $(i, j, k) \in I \times J \times K$. Another way to formulate the AP3 is with permutations. There are n^3 cost elements. The optimal solution consists of the n smallest cost elements, such that the constraints are not violated. The constraints are enforced if one assigns to each set I, J, and K, the numbers $1, 2, \ldots, n$ and none of the chosen triplets (i, j, k) is allowed to have the same value for indices i, j, and k as another. The permutation-based formulation for the AP3 is

$$\min_{p,q \in \pi_N} \sum_{i=1}^{n} c_{ip(i)q(i)},$$

where π_N denotes the set of all permutations of the set of integers $N = \{1, 2, \ldots, n\}$.

7.4.1.2 GRASP construction. The construction phase selects n triplets, one at a time, to form a three-index assignment S. The usual random choice in the interval $[0, 1]$ for the RCL parameter α is made at each iteration. The value remains constant during the entire construction phase. Construction begins with an empty solution S. The initial set C of candidate triplets consists of the set of all triplets. Let $\underline{c}$ and $\overline{c}$ denote, respectively, the values of the smallest and largest cost triplets in C. All triplets (i, j, k) in the candidate set C having cost $c_{ijk} \leq \underline{c} + \alpha(\overline{c} - \underline{c})$ are placed in

the RCL. Triplet $(i_p, j_p, k_p) \in C'$ is chosen at random and is added to the solution, i.e. $S = S \cup \{(i_p, j_p, k_p)\}$. Once (i_p, j_p, k_p) is selected, any triplet $(i, j, k) \in C$ such that $i = i_p$ or $j = j_p$ or $k = k_p$ is removed from C. After $n-1$ triplets have been selected, the set C of candidate triplets contains one last triplet which is added to S, thus completing the construction phase.

7.4.1.3 Local search. If the solution of the AP3 is represented by a pair of permutations (p, q), then the solution space consists of all $(n!)^2$ possible combinations of permutations. If p is a permutation vector, then a 2-exchange permutation of p is a permutation vector that results from swapping two elements in p. In the 2-exchange neighborhood scheme used in this local search, the neighborhood of a solution (p, q) consists of all 2-exchange permutations of p plus all 2-exchange permutations of q. In the local search, the cost of each neighbor solution is compared with the cost of the current solution. If the cost of the neighbor is lower, then the solution is updated, the search is halted, and a search in the new neighborhood is initialized. The local search ends when no neighbor of the current solution has a lower cost than the current solution.

7.4.1.4 Path-relinking. A solution of AP3 can be represented by two permutation arrays of numbers $1, 2, \ldots, n$ in sets J and K, respectively, as follows:

$$S = \{(p_1^S, p_2^S, \ldots, p_n^S), (q_1^S, q_2^S, \ldots, q_n^S)\}.$$

Path-relinking is done between an initial solution

$$S = \{(p_1^S, p_2^S, \ldots, p_n^S), (q_1^S, q_2^S, \ldots, q_n^S)\}$$

and a guiding solution

$$T = \{(p_1^T, p_2^T, \ldots, p_n^T), (q_1^T, q_2^T, \ldots, q_n^T)\}.$$

Let the difference between S and T be defined by the two sets of indices

$$\delta_p^{S,T} = \{i = 1, \ldots, n \mid p_i^S \neq p_i^T\},$$
$$\delta_q^{S,T} = \{i = 1, \ldots, n \mid q_i^S \neq q_i^T\}.$$

During a path-relinking move, a permutation π (p or q) array in S, given by

$$(\ldots, \pi_i^S, \pi_{i+1}^S, \ldots, \pi_{j-1}^S, \pi_j^S, \ldots),$$

is replaced by a permutation array

$$(\ldots, \pi_j^S, \pi_{i+1}^S, \ldots, \pi_{j-1}^S, \pi_i^S, \ldots),$$

by exchanging permutation elements π_i^S and π_j^S, where $i \in \delta_\pi^{S,T}$ and $j \in \{1, 2, \ldots, n\}$ are such that $\pi_j^T = \pi_i^S$.

Table 7.1 Estimated exponential distribution parameters μ and λ obtained with 200 independent runs of a sequential GRASP with path-relinking on AP3 instances B-S 20.1, B-S 22.1, B-S 24.1, and B-S 26.1, with target values 7, 8, 7, and 8, respectively.

	estimated parameter		
Problem	μ	λ	$\lvert\mu\rvert/\lambda$
B-S 20.1	-26.46	1223.80	.021
B-S 22.1	-135.12	3085.32	.043
B-S 24.1	-16.76	4004.11	.004
B-S 26.1	32.12	2255.55	.014
average			**.020**

Table 7.2 Speedups for multiple-walk independent-thread implementations of GRASP with path-relinking on instances B-S 20.1, B-S 22.1, B-S 24.1, and B-S 26.1, with target values 7, 8, 7, and 8, respectively. Speedups are computed with the average of 60 runs.

	number of processors							
	2		4		8		16	
Problem	speedup	effic.	speedup	effic.	speedup	effic.	speedup	effic.
B-S 20.1	1.67	0.84	3.34	0.84	6.22	0.78	10.82	0.68
B-S 22.1	2.25	1.13	4.57	1.14	9.01	1.13	14.37	0.90
B-S 24.1	1.71	0.86	4.00	1.00	7.87	0.98	12.19	0.76
B-S 26.1	2.11	1.06	3.89	0.97	6.10	0.76	11.49	0.72
average	**1.94**	**0.97**	**3.95**	**0.99**	**7.3**	**0.91**	**12.21**	**0.77**

7.4.1.5 Parallel independent-thread GRASP with path-relinking for AP3. We study the parallel efficiency of the multiple-walk independent-thread GRASP with path-relinking on AP3 instances B-S 20.1, B-S 22.1, B-S 24.1, and B-S 26.1 of Balas and Saltzman (1991) using 7, 8, 7, and 8 as target solution values, respectively. Table 7.1 shows the estimated exponential distribution parameters for the multiple-walk independent-thread GRASP with path-relinking strategy obtained from 200 independent runs of a sequential variant of the algorithm. In addition to the sequential variant, 60 independent runs of 2-, 4-, 8-, and 16-thread variants were run on the four test problems. Average speedups were computed dividing the sum of the execution times of the independent parallel program executing on one processor by the sum of the execution times of the parallel program on 2, 4, 8, and 16 processors, for 60 runs. The execution times of the independent parallel program executing on one processor and the execution times of the sequential program are approximately the same. The average speedups can be seen in Table 7.2 and Figure 7.9.

7.4.1.6 Parallel cooperative-thread GRASP with path-relinking for AP3. We now study the multiple-walk cooperative-thread strategy for GRASP with

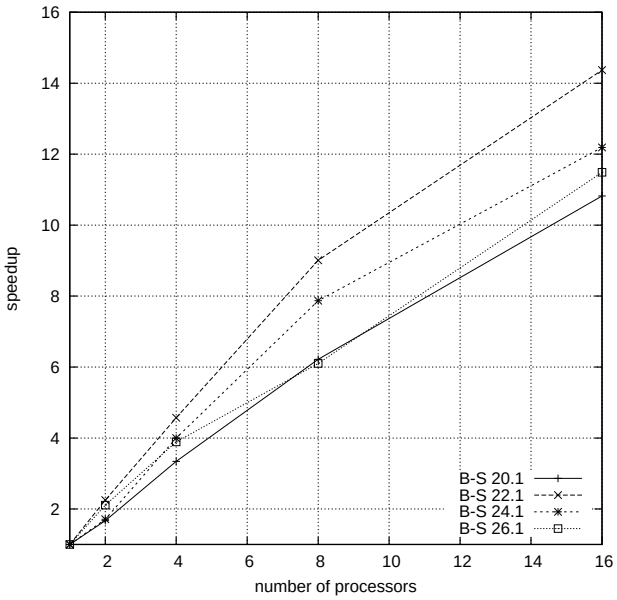


Figure 7.9 Average speedups on 2, 4, 8, and 16 processors for multiple-walk independent-thread parallel GRASP with path-relinking on AP3 instances `B-S 20.1`, `B-S 22.1`, `B-S 24.1`, and `B-S 26.1`.

Table 7.3 Speedups for multiple-walk cooperative-thread implementations of GRASP with path-relinking on instances `B-S 20.1`, `B-S 22.1`, `B-S 24.1`, and `B-S 26.1`, with target values 7, 8, 7, and 8, respectively. Average speedups were computed over 60 runs.

	number of processors							
	2		4		8		16	
Problem	speedup	effic.	speedup	effic.	speedup	effic.	speedup	effic.
`B-S 20.1`	1.56	0.78	3.47	0.88	7.37	0.92	14.36	0.90
`B-S 22.1`	1.64	0.82	4.22	1.06	8.83	1.10	18.78	1.04
`B-S 24.1`	2.16	1.10	4.00	1.00	9.38	1.17	19.29	1.21
`B-S 26.1`	2.16	1.08	5.30	1.33	9.55	1.19	16.00	1.00
average	**1.88**	**0.95**	**4.24**	**1.07**	**8.78**	**1.10**	**17.10**	**1.04**

path-relinking on the AP3. As with the independent-thread GRASP with path-relinking strategy, the target solution values 7, 8, 7, and 8 were used for instances `B-S 20.1`, `B-S 22.1`, `B-S 24.1`, and `B-S 26.1`, respectively. Table 7.3 and Figure 7.10 show super-linear speedups on instances `B-S 22.1`, `B-S 24.1`, and `B-S 26.1` and about 90% efficiency for `B-S 20.1`. Super-linear speedups are possible because good elite solutions are shared among the threads and are combined with GRASP solutions, whereas they would not be combined in an independent-thread implementation.

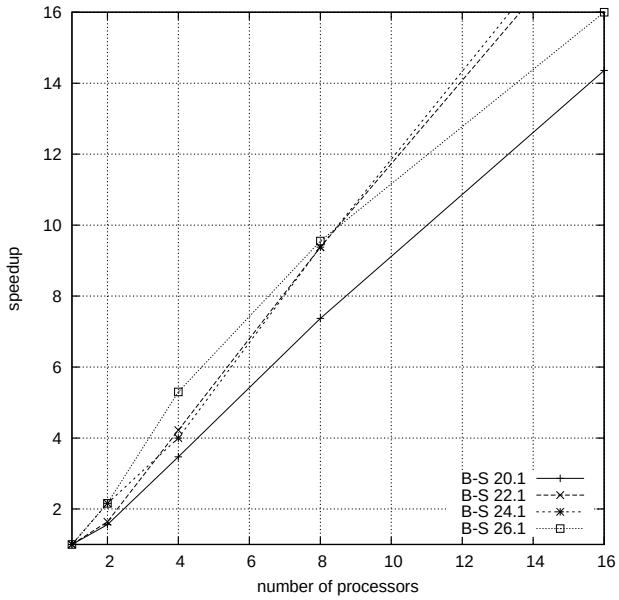


Figure 7.10 Average speedups on 2, 4, 8, and 16 processors for multiple-walk cooperative-thread parallel GRASP with path-relinking on AP3 instances `B-S 20.1`, `B-S 22.1`, `B-S 24.1`, and `B-S 26.1`.

Figure 7.11 compares average speedup of the two implementations tested in this section, namely the multiple-walk independent-thread and multiple-walk cooperative-thread GRASP with path-relinking implementations using target solution values 7, 8, 7, and 8, on the same instances. The figure shows that the cooperative variant of GRASP with path-relinking achieves the best parallelization.

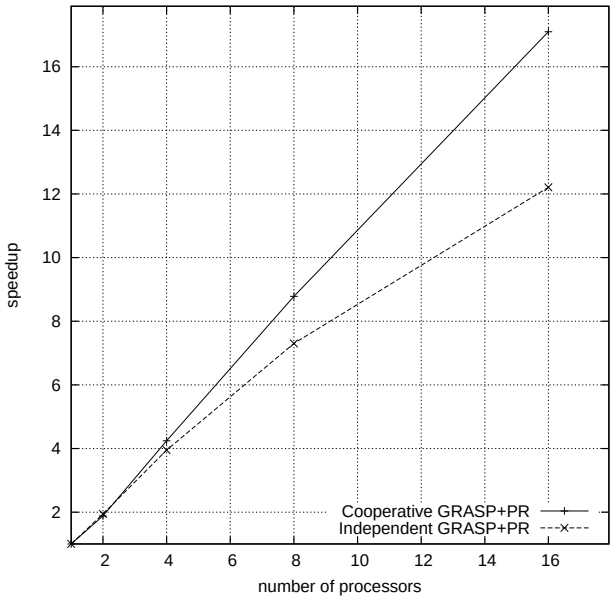


Figure 7.11 Average speedups on 2, 4, 8, and 16 processors for the parallel algorithms tested on instances of AP3: multiple-walk independent-thread GRASP with path-relinking and multiple-walk cooperative-thread GRASP with path-relinking.

7.4.2 Job shop scheduling

7.4.2.1 Problem formulation. The job shop scheduling problem (JSP) is an NP-hard (Lenstra and Rinnooy Kan, 1979) combinatorial optimization problem that has long challenged researchers. It consists in processing a finite set of jobs on a finite set of machines. Each job is required to complete a set of operations in a fixed order. Each operation is processed on a specific machine for a fixed duration. Each machine can process at most one job at a time and once a job initiates processing on a given machine, it must complete processing on that machine without interruption. A schedule is a mapping of operations to time slots on the machines. The makespan is the maximum completion time of the jobs. The objective of the JSP is to find a schedule that minimizes the makespan.

A feasible solution of the JSP can be built from a permutation of the set of jobs $\mathcal{J}$ on each of the machines in the set $\mathcal{M}$, observing the precedence constraints, the restriction that a machine can process only one operation at a time, and requiring that once started, processing of an operation cannot be interrupted until its completion. Since each set of feasible permutations has a corresponding schedule, the objective of the JSP is to find, among the feasible permutations, the one with the smallest makespan.

7.4.2.2 GRASP construction. Consider the GRASP construction phase for the JSP, proposed in Binato et al. (2002) and Aiex et al. (2003), where a single oper-

ation is the building block of the construction phase. A feasible schedule is built by scheduling individual operations, one at a time, until all operations have been scheduled.

While constructing a feasible schedule, not all operations can be selected at a given stage of the construction. An operation σ_k^j can only be scheduled if all prior operations of job j have already been scheduled. Therefore, at each construction phase iteration, at most $|\mathcal{J}|$ operations are candidates to be scheduled. Let this set of candidate operations be denoted by O_c and the set of already scheduled operations by O_s. Denote the value of the greedy function for candidate operation σ_k^j by $h(\sigma_k^j)$.

The greedy choice is to next schedule operation $\underline{\sigma}_k^j = \text{argmin}(h(\sigma_k^j) \mid \sigma_k^j \in O_c)$. Let $\overline{\sigma}_k^j = \text{argmax}(h(\sigma_k^j) \mid \sigma_k^j \in O_c)$, $\underline{h} = h(\underline{\sigma}_k^j)$, and $\overline{h} = h(\overline{\sigma}_k^j)$. Then, the GRASP restricted candidate list (RCL) is defined as

$$\text{RCL} = \{\sigma_k^j \in O_c \mid \underline{h} \leq h(\sigma_k^j) \leq \underline{h} + \alpha(\overline{h} - \underline{h})\},$$

where α is a parameter such that $0 \leq \alpha \leq 1$.

A typical iteration of the GRASP construction is summarized as follows: a partial schedule (which is initially empty) is on hand, the next operation to be scheduled is selected from the RCL and is added to the partial schedule, resulting in a new partial schedule. The selected operation is inserted into the earliest available feasible time slot on machine $\mathcal{M}_{\sigma_k^j}$. Construction ends when the partial schedule is complete, i.e. all operations have been scheduled.

The algorithm uses two greedy functions. Even numbered iterations use a greedy function based on the makespan resulting from the inclusion of operation σ_k^j to the already-scheduled operations, i.e. $h(\sigma_k^j) = C_{max}$ for $O = \{O_s \cup \sigma_k^j\}$. On odd numbered iterations, solutions are constructed by favoring operations from jobs having long remaining processing times. The greedy function used is given by $h(\sigma_k^j) = -\sum_{\sigma_l^j \notin O_s} p_l^j$, which measures the remaining processing time for job j. The use of two different greedy functions produce a greater diversity of initial solutions to be used by the local search.

7.4.2.3 Local search. To attempt to decrease the makespan of the solution produced in the construction phase, we employ the 2-exchange local search used in (Aiex et al., 2003; Binato et al., 2002; Taillard, 1991), based on the disjunctive graph model of Roy and Sussmann (1964). We refer the reader to Aiex et al. (2003) and Binato et al. (2002) for a description of the implementation of the local search procedure.

7.4.2.4 Path-relinking. Path-relinking for job shop scheduling is similar to path-relinking for three-index assignment. Where in the case of three-index assignment each solution is represented by two permutation arrays, in the job shop scheduling problem, each solution is made up of $|\mathcal{M}|$ permutation arrays of numbers $1, 2, \ldots, |\mathcal{J}|$.

7.4.2.5 Parallel independent-thread GRASP with path-relinking for JSP. We study the efficiency of the multiple-walk independent-thread GRASP with path-relinking on JSP instances `abz6`, `mt10`, `orb5`, and `la21` of ORLib (Beasley, 1990)

Table 7.4 Estimated exponential distribution parameters μ and λ obtained with 200 independent runs of a sequential GRASP with path-relinking on JSP instances abz6, mt10, orb5, and la21, with target values 943, 938, 895, and 1100, respectively.

	estimated parameter		
Problem	μ	λ	$\lvert\mu\rvert/\lambda$
abz6	47.67	756.56	.06
mt10	305.27	524.23	.58
orb5	130.12	395.41	.32
la21	175.20	407.73	.42
average			**.34**

Table 7.5 Speedups for multiple-walk independent-thread implementations of GRASP with path-relinking on instances abz6, mt10, orb5, and la21, with target values 943, 938, 895, and 1100, respectively. Speedups are computed with the average of 60 runs.

	number of processors							
	2		4		8		16	
Problem	speedup	effic.	speedup	effic.	speedup	effic.	speedup	effic.
abz6	2.00	1.00	3.36	0.84	6.44	0.81	10.51	0.66
mt10	1.57	0.79	2.12	0.53	3.03	0.39	4.05	0.25
orb5	1.95	0.98	2.97	0.74	3.99	0.50	5.36	0.34
la21	1.64	0.82	2.25	0.56	3.14	0.39	3.72	0.23
average	**1.79**	**0.90**	**2.67**	**0.67**	**4.15**	**0.52**	**5.91**	**0.37**

using 943, 938, 895, and 1100 as target solution values, respectively. Table 7.4 shows the estimated exponential distribution parameters for the multiple-walk independent-thread GRASP with path-relinking strategy obtained from 200 independent runs of a sequential variant of the algorithm. In addition to the sequential variant, 60 independent runs of 2-, 4-, 8-, and 16-thread variants were run on the four test problems. As before, average speedups were computed dividing the sum of the execution times of the independent parallel program executing on one processor by the sum of the execution times of the parallel program on 2, 4, 8, and 16 processors, for 60 runs. The average speedups can be seen in Table 7.5 and Figure 7.12.

Compared to the efficiencies observed on the AP3 instances, those for these instances of the JSP were much worse. While with 16 processors average speedups of 12.2 were computed for the AP3, average speedups of only 5.9 were computed for the JSP. This is consistent with the $|\mu|/\lambda$ values, which were on average .34 for the JSP, and 0.02 for the AP3.

7.4.2.6 Parallel cooperative-thread GRASP with path-relinking for JSP. We now study the multiple-walk cooperative-thread strategy for GRASP with path-relinking on the JSP. As with the independent-thread GRASP with path-relinking

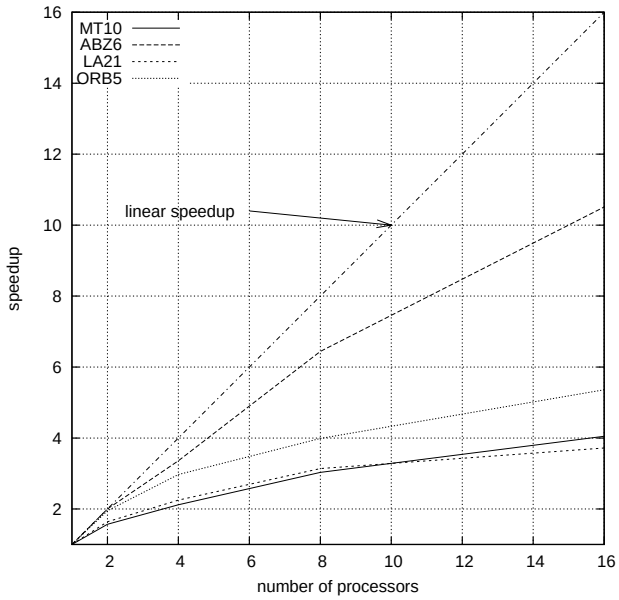


Figure 7.12 Average speedups on 2, 4, 8, and 16 processors for multiple-walk independent-thread parallel GRASP with path-relinking on JSP instances `abz6`, `mt10`, `orb5`, and `la21`.

Table 7.6 Speedups for multiple-walk cooperative-thread implementations of GRASP with path-relinking on instances `abz6`, `mt10`, `orb5`, and `la21`, with target values 943, 938, 895, and 1100, respectively. Average speedups were computed over 60 runs.

	number of processors							
	2		4		8		16	
Problem	speedup	effic.	speedup	effic.	speedup	effic.	speedup	effic.
`abz6`	2.40	1.20	4.21	1.05	11.43	1.43	23.58	1.47
`mt10`	1.75	0.88	4.58	1.15	8.36	1.05	16.97	1.06
`orb5`	2.10	1.05	4.91	1.23	8.89	1.11	15.76	0.99
`la21`	2.23	1.12	4.47	1.12	7.54	0.94	11.41	0.71
average	**2.12**	**1.06**	**4.54**	**1.14**	**9.05**	**1.13**	**16.93**	**1.06**

strategy, the target solution values 943, 938, 895, and 1100 were used for instances `abz6`, `mt10`, `orb5`, and `la21`, respectively. Table 7.6 and Figure 7.13 show super-linear speedups on instances `abz6` and `mt10`, linear speedup on `orb5` and about 70% efficiency for `la21`. As before, super-linear speedups are possible because good elite solutions are shared among the threads and these elite solutions are combined with GRASP solutions whereas they would not be combined in an independent-thread implementation.

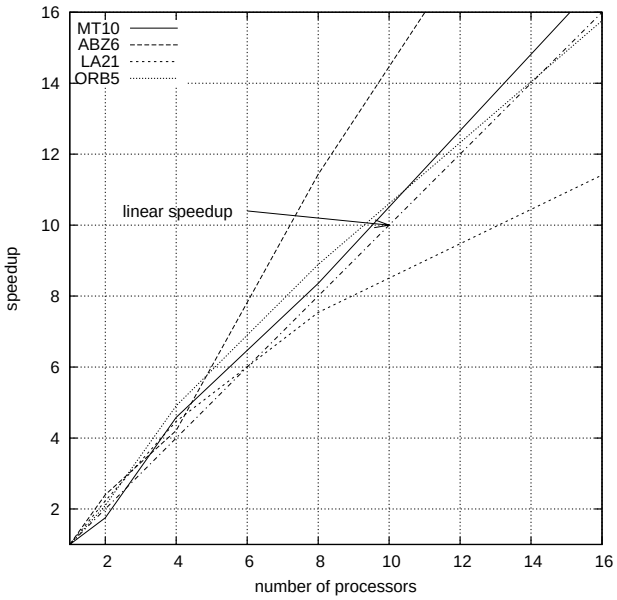


Figure 7.13 Average speedups on 2, 4, 8, and 16 processors for multiple-walk cooperative-thread parallel GRASP with path-relinking on JSP instances `abz6`, `mt10`, `orb5`, and `la21`.

Figure 7.14 compares the average speedup of the two implementations tested in this section, namely implementations of the multiple-walk independent-thread and multiple-walk cooperative-thread GRASP with path-relinking using target solution values 943, 938, 895, and 1100, on instances `abz6`, `mt10`, `orb5`, and `la21`, respectively.

The figure shows that the cooperative variant of GRASP with path-relinking achieves the best parallelization.

7.4.3 2-path network design problem

7.4.3.1 Problem formulation. Let $G = (V, E)$ be a connected graph, where V is the set of nodes and E is the set of edges. A k-path between nodes $s, t \in V$ is a sequence of at most k edges connecting them. Given a non-negative weight function $w : E \rightarrow R_+$ associated with the edges of G and a set D of pairs of origin-destination nodes, the *2-path network design problem* (2PNDP) consists of finding a minimum weighted subset of edges $E' \subseteq E$ containing a 2-path between every origin-destination pair.

Applications of 2PNDP can be found in the design of communications networks, in which paths with few edges are sought to enforce high reliability and small delays. 2PNDP was shown to be NP-hard by Dahl and Johannessen (2004).

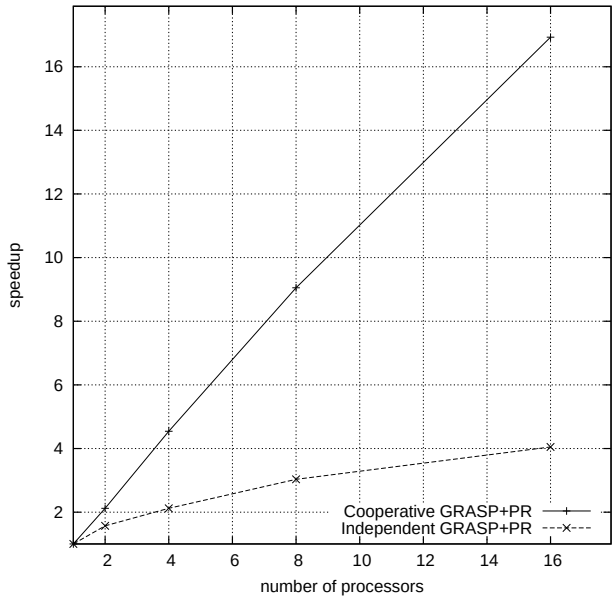


Figure 7.14 Average speedups on 2, 4, 8, and 16 processors for the parallel algorithms tested on instances of JSP: multiple-walk independent-thread GRASP with path-relinking and multiple-walk cooperative-thread GRASP with path-relinking.

7.4.3.2 GRASP construction. The construction of a new solution begins by the initialization of modified edge weights with the original edge weights. Each iteration of the construction phase starts by the random selection of an origin-destination pair still in D. A shortest 2-path between the extremities of this pair is computed, using the modified edge weights. The weights of the edges in this 2-path are set to zero until the end of the construction procedure, the origin-destination pair is removed from D, and a new iteration resumes. The construction phase stops when 2-paths have been computed for all origin-destination pairs.

7.4.3.3 Local search. The local search phase seeks to improve each solution built in the construction phase. Each solution may be viewed as a set of 2-paths, one for each origin-destination pair in D. To introduce some diversity by driving different applications of the local search to different local optima, the origin-destination pairs are investigated at each GRASP iteration in a circular order defined by a different random permutation of their original indices.

Each 2-path in the current solution is tentatively eliminated. The weights of the edges used by other 2-paths are temporarily set to zero, while those which are not used by other 2-paths in the current solution are restored to their original values. A new shortest 2-path between the extremities of the origin-destination pair under investigation is computed, using the modified weights. If the new 2-path improves the current solution, then the latter is modified; otherwise the previous 2-path is restored.

The search stops if the current solution was not improved after a sequence of $|D|$ iterations along which all 2-paths have been investigated. Otherwise, the next 2-path in the current solution is investigated for substitution and a new iteration resumes.

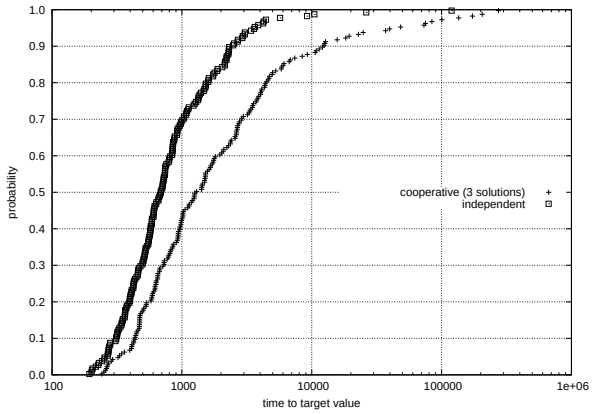
7.4.3.4 Path-relinking. A solution to 2PNDP is represented as a set of 2-paths connecting each origin-destination pair. Path-relinking starts by determining all origin-destination pairs whose associated 2-paths are different in the starting and guiding solutions. These computations amount to determining a set of moves which should be applied to the initial solution to reach the guiding one. Each move is characterized by a pair of 2-paths, one to be inserted and the other to be eliminated from the current solution.

7.4.3.5 Parallel implementations of GRASP with path-relinking for 2PNDP. As for problems AP3 and JSP, in the case of the independent-thread parallel implementation of GRASP with path-relinking for 2PNDP, each processor has a copy of the sequential algorithm, a copy of the data, and its own pool of elite solutions. One processor acts as the master, reading and distributing the problem data, generating the seeds which will be used by the pseudo-random number generators at each processor, distributing the iterations, and collecting the best solution found by each processor. All the p available processors perform GRASP iterations.

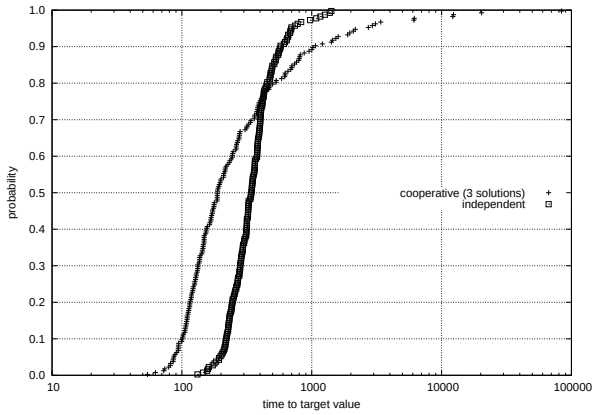
However, in the case of the cooperative-thread parallel implementation of GRASP with path-relinking for 2PNDP, the master handles a centralized pool of elite solutions, collecting and distributing them upon request (recall that in the case of AP3 and JSP each processor had its own pool of elite solutions). The $p-1$ slaves exchange the elite solutions found along their search trajectories. In the proposed implementation for 2PNDP, each slave may send up to three different solutions to the master at each iteration: the solution obtained by local search, and the solutions w^1 and w^2 obtained by forward and backward path-relinking (Resende and Ribeiro, 2005) between the same pair of starting and guiding solutions, respectively.

7.4.3.6 Computational results. The results illustrated in this section concern an instance with 100 nodes, 4950 edges, and 1000 origin-destination pairs. We use the methodology proposed in Aiex et al. (2002) to assess experimentally the behavior of randomized algorithms. This approach is based on plots showing empirical distributions of the random variable *time to target solution value*. To plot the empirical distribution, we fix a solution target value and run each algorithm 200 times, recording the running time when a solution with cost at least as good as the target value is found. For each algorithm, we associate with the i-th sorted running time t_i a probability $p_i = (i - \frac{1}{2})/200$ and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 200$.

Results obtained for both the independent-thread and the cooperative-thread parallel implementations of GRASP with path-relinking on the above instance with the target value set at 683 are reported in Figure 7.15. The cooperative implementation is already faster than the independent one for eight processors. For fewer processors the independent implementation is naturally faster, since it employs all p processors in



(a)



(b)

Figure 7.15 Running times for 200 runs of (a) the multiple-walk independent-thread and (b) the multiple-walk cooperative-thread implementations of GRASP with path-relinking using two processors and with the target solution value set at 683.

the search (while only $p-1$ slave processors take part effectively in the computations performed by the cooperative implementation).

Three different strategies were investigated to further improve the performance of the cooperative-thread implementation, by reducing the cost of the communication between the master and the slaves when the number of processors increases:

(1) Each send operation is broken in two parts. First, the slave sends only the cost of the solution to the master. If this solution is better than the worst solution in the pool, then the full solution is sent. The number of messages increases, but most of them will be very small ones with light memory requirements.

(2) Only one solution is sent to the pool at each GRASP iteration.

(3) A distributed implementation, in which each slave handles its own pool of elite solutions. Every time a processor finds a new elite solution, the latter is broadcast to the others.

Comparative results for these three strategies on the same problem instance are plotted in Figure 7.16. The first strategy outperformed all others.

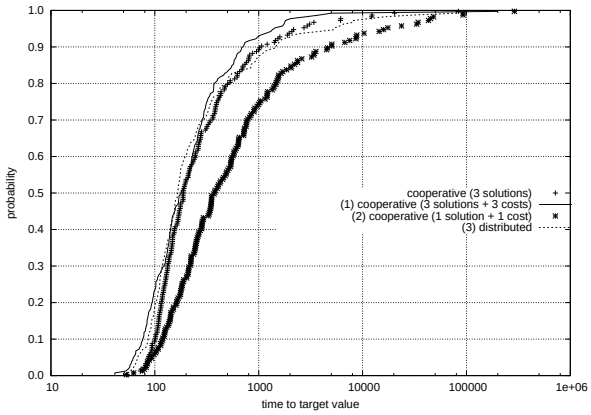


Figure 7.16 Strategies for improving the performance of the centralized multiple-walk cooperative-thread implementation on eight processors.

Table 7.7 shows the average computation times and the best solutions found over ten runs of each strategy when the total number of GRASP iterations is set at 3200. There is a clear degradation in solution quality for the independent-thread strategy when the number of processors increases. As fewer iterations are performed by each processor, the pool of elite solutions gets poorer with the increase in the number of processors. Since the processors do not communicate, the overall solution quality is worse. In the case of the cooperative strategy, the information shared by the processors guarantees the high quality of the solutions in the pool. The cooperative implementation is more robust. Very good solutions are obtained with no degradation in quality and significant speedups.

7.5 CONCLUSION

Metaheuristics, such as GRASP, have found their way into the standard toolkit of combinatorial optimization methods. Parallel computers have increasingly found their way into metaheuristics.

In this chapter, we surveyed work on the parallelization of GRASP. We first showed that the random variable *time to target solution value* for GRASP heuristics fits a two-parameter (shifted) exponential distribution. Under the mild assumption that the product of the number of processors by the shift in the distribution is small compared to the standard deviation of the distribution, linear speedups can be expected in parallel

Table 7.7 Average times and best solutions over ten runs for 2PNDP.

	independent		cooperative	
processors	best value	avg. time (s)	best value	avg. time (s)
1	673	1310.1	—	—
2	676	686.8	676	1380.9
4	680	332.7	673	464.1
8	687	164.1	676	200.9
16	692	81.7	674	97.5
32	702	41.3	678	74.6

multiple-walk independent-thread implementations. We illustrated with an application to the maximum satisfiability problem a case where this occurs.

Path-relinking has been increasingly used to introduce memory in the otherwise memoryless original GRASP procedure. The hydridization of GRASP and path-relinking has led to some effective multiple-walk cooperative-thread implementations. Collaboration between the threads is usually achieved by sharing elite solutions, either in a single centralized pool or in distributed pools. In some of these implementations, super-linear speedups are achieved even for cases where little speedup occurs in multiple-walk independent-thread variants.

Parallel cooperative implementations of metaheuristics lead to significant speedups, smaller computation times, and more robust algorithms. However, they demand more programming efforts and implementation skills. The three applications described in this survey illustrate the strategies and programming skills involved in the development of robust and efficient parallel cooperative implementations of GRASP.

Bibliography

R.M. Aiex. *Uma investigação experimental da distribuição de probabilidade de tempo de solução em heurísticas GRASP e sua aplicação na análise de implementações paralelas*. PhD thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, 2002.

R.M. Aiex, S. Binato, and M.G.C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29:393–430, 2003.

R.M. Aiex and M.G.C. Resende. Parallel strategies for GRASP with path-relinking. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as real problem solvers*. Springer, 2005. To appear.

R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with path relinking for three-index assignment. *INFORMS Journal on Computing*, 17, 2005. In press.

R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8:343–373, 2002.

A. Alvim and C.C. Ribeiro. Balanceamento de carga na paralelização da metaheurística GRASP. In *X Simpósio Brasileiro de Arquiteturas de Computadores*, pages 279–282. Sociedade Brasileira de Computação, 1998.

A.C.F. Alvim. Estratégias de paralelização da metaheurística GRASP. Master's thesis, Departamento de Informática, PUC-Rio, Rio de Janeiro, RJ 22453-900 Brazil, April 1998.

E. Balas and M.J. Saltzman. An algorithm for the three-index assignment problem. *Oper. Res.*, 39:150–161, 1991.

R. Battiti and G. Tecchiolli. Parallel biased search for combinatorial optimization: Genetic algorithms and TABU. *Microprocessors and Microsystems*, 16:351–367, 1992.

J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

S. Binato, H. Faria Jr., and M.G.C. Resende. Greedy randomized adaptive path relinking. In J.P. Sousa, editor, *Proceedings of the IV Metaheuristics International Conference*, pages 393–397, 2001.

S. Binato, W.J. Hery, D.M. Loewenstern, and M.G.C. Resende. A GRASP for job shop scheduling. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys on Metaheuristics*, pages 58–79. Kluwer Academic Publishers, 2002.

R. Burkard, S. Karisch, and F. Rendl. QAPLIB – A quadratic assignment problem library. *European Journal of Operations Research*, 55:115–119, 1991.

S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38:50–58, 2001.

J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. *Graphical Methods for Data Analysis*. Chapman & Hall, 1983.

V.-D. Cung, S.L. Martins, C.C. Ribeiro, and C. Roucairol. Strategies for the parallel implementation of metaheuristics. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 263–308. Kluwer Academic Publishers, 2002.

G. Dahl and B. Johannessen. The 2-path network design problem. *Networks*, 43: 190–199, 2004.

N. Dodd. Slow annealing versus multiple fast annealing runs: An empirical investigation. *Parallel Computing*, 16:269–272, 1990.

L.M.A. Drummond, L.S. Vianna, M.B. Silva, and L.S. Ochi. Distributed parallel metaheuristics based on GRASP and VNS for solving the traveling purchaser problem. In *Proceedings of the Ninth International Conference on Parallel and Distributed Systems – ICPADS'02*, pages 1–7. IEEE, 2002.

S. Duni, P.M. Pardalos, and M.G.C. Resende. Parallel metaheuristics for combinatorial optimization. In R. Corrêa, I. Dutra, M. Fiallos, and F. Gomes, editors, *Models for Parallel and Distributed Computation – Theory, Algorithmic Techniques and Applications*, pages 179–206. Kluwer Academic Publishers, 2002.

H.M.M. Ten Eikelder, M.G.A. Verhoeven, T.W.M. Vossen, and E.H.L. Aarts. A probabilistic analysis of local search. In I.H. Osman and J.P. Kelly, editors, *Metaheuristics: Theory & applications*, pages 605–618. Kluwer Academic Publishers, 1996.

H. Faria Jr., S. Binato, M.G.C. Resende, and D.J. Falcão. Transmission network design by a greedy randomized adaptive path relinking approach. *IEEE Transactions on Power Systems*, 20(1), 2005. In press.

T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.

P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

P. Festa and M.G.C. Resende. An annotated bibliography of GRASP. Technical Report TD-5WYSEW, AT&T Labs Research, Florham Park, NJ 07932, February 2004.

A.M. Frieze. Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research*, 13:161–164, 1983.

M.R. Garey and D.S. Johnson. *Computers and intractability - A guide to the theory of NP-completeness*. W.H. Freeman and Company, 1979.

A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, and V. Sunderam. *PVM: Parallel virtual machine, A user's guide and tutorial for networked parallel computing*. Scientific and Engineering Computation. MIT Press, Cambridge, MA, 1994.

F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

F. Glover. Multi-start and strategic oscillation methods – Principles to exploit adaptive memory. In M. Laguna and J.L. Gonzáles-Velarde, editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pages 1–24. Kluwer, 2000.

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. Technical report, Graduate School of Business and Administration, University of Colorado, Boulder, CO 80309-0419, 2000.

H. Hoos and T. Stützle. Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. *Artificial Intelligence*, 112:213–232, 1999.

Kendall Square Research. *KSR Parallel Programming*. 170 Tracer Lane, Waltham, MA, February 1992.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

J. K. Lenstra and A. H. G. Rinnooy Kan. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.

Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994.

S.L. Martins, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Greedy randomized adaptive search procedures for the Steiner problem in graphs. In P.M. Pardalos, S. Rajasejaran, and J. Rolim, editors, *Randomization methods in algorithmic design*, volume 43 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 133–145. American Mathematical Society, 1999.

S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P.M. Pardalos. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization*, pages 267–283, 2000.

S.L. Martins, C.C. Ribeiro, and I. Rosseti. Applications and parallel implementations of metaheuristics in network design and routing. *Lecture Notes in Computer Science*, 3285:205–213, 2004.

S.L. Martins, C.C. Ribeiro, and M.C. Souza. A parallel GRASP for the Steiner problem in graphs. In A. Ferreira and J. Rolim, editors, *Proceedings of IRREGULAR'98 – 5th International Symposium on Solving Irregularly Structured Problems in Parallel*, volume 1457 of *Lecture Notes in Computer Science*, pages 285–297. Springer-Verlag, 1998.

R.A. Murphey, P.M. Pardalos, and L.S. Pitsoulis. A parallel GRASP for the data association multidimensional assignment problem. In P.M. Pardalos, editor, *Parallel processing of discrete problems*, volume 106 of *The IMA Volumes in Mathematics and Its Applications*, pages 159–180. Springer-Verlag, 1998.

L.J. Osborne and B.E. Gillett. A comparison of two simulated annealing algorithms applied to the directed Steiner problem on networks. *ORSA Journal on Computing*, 3:213–225, 1991.

P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP implementation for the quadratic assignment problem. In A. Ferreira and J. Rolim, editors, *Parallel Algorithms for Irregularly Structured Problems – Irregular'94*, pages 115–133. Kluwer Academic Publishers, 1995.

P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP for MAX-SAT problems. *Lecture Notes in Computer Science*, 1184:575–585, 1996.

W.P. Pierskalla. The tri-subsitution method for the three-multidimensional assignment problem. *CORS Journal*, 5:71–81, 1967.

M. Prais and C.C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12: 164–176, 2000.

M.G.C. Resende. Computing approximate solutions of the maximum covering problem using GRASP. *J. of Heuristics*, 4:161–171, 1998.

M.G.C. Resende, T.A. Feo, and S.H. Smith. Algorithm 787: Fortran subroutines for approximate solution of maximum independent set problems using GRASP. *ACM Trans. Math. Software*, 24:386–394, 1998.

M.G.C. Resende, P.M. Pardalos, and Y. Li. Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software*, 22:104–118, 1996.

M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Fortran subroutines for computing approximate solutions of MAX-SAT problems using GRASP. *Discrete Applied Mathematics*, 100:95–113, 2000.

M.G.C. Resende and C.C. Ribeiro. A GRASP for graph planarization. *Networks*, 29: 173–189, 1997.

M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2002.

M.G.C. Resende and C.C. Ribeiro. A GRASP with path-relinking for private virtual circuit routing. *Networks*, 41:104–114, 2003.

M.G.C. Resende and C.C. Ribeiro. GRASP with path-relinking: Recent advances and applications. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as real problem solvers*. Springer, 2005. To appear.

M.G.C. Resende and R.F. Werneck. A hybrid heuristic for the *p*-median problem. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ, 2002.

C.C. Ribeiro and M.G.C. Resende. Algorithm 797: Fortran subroutines for approximate solution of graph planarization problems using GRASP. *ACM Transactions on Mathematical Software*, 25:341–352, 1999.

C.C. Ribeiro and I. Rosseti. A parallel GRASP for the 2-path network design problem. *Lecture Notes in Computer Science*, 2004:922–926, 2002.

C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14:228–246, 2002.

I. Rosseti. *Heurísticas para o problema de síntese de redes a 2-caminhos*. PhD thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, July 2003.

B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives, 1964.

B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 337–343, Seattle, 1994. MIT Press.

M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI – The complete reference, Volume 1 – The MPI Core*. The MIT Press, 1998.

E.D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.

M.G.A. Verhoeven and E.H.L. Aarts. Parallel local search. *Journal of Heuristics*, 1: 43–66, 1995.

8 PARALLEL STRATEGIES FOR GRASP WITH PATH-RELINKING

Renata M. Aiex[1] and Mauricio G. C. Resende[2]

[1]Department of Computer Science
Catholic University of Rio de Janeiro
Rio de Janeiro, RJ 22453-900 Brazil
rma@inf.puc-rio.br

[2]Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA.
mgcr@research.att.com

Abstract: A Greedy Randomized Adaptive Search Procedure (GRASP) is a metaheuristic for combinatorial optimization. It usually consists of a construction procedure based on a greedy randomized algorithm and a local search. Path-relinking is an intensification strategy that explores trajectories that connect high quality solutions. We analyze two parallel strategies for GRASP with path-relinking and propose a criterion to predict parallel speedup based on experiments with a sequential implementation of the algorithm. Independent and cooperative parallel strategies are described and implemented for the 3-index assignment problem and the job-shop scheduling problem. The computational results for independent parallel strategies are shown to qualitatively behave as predicted by the criterion.
Keywords: Combinatorial optimization, job shop scheduling, 3-index assignment, local search, GRASP, path-relinking, parallel algorithm.

8.1 INTRODUCTION

GRASP (Greedy Randomized Adaptive Search Procedure) is a metaheuristic for combinatorial optimization (Feo and Resende, 1989; 1995; Festa and Resende, 2002; Resende and Ribeiro, 2002). A GRASP is an iterative process, where each iteration usually consists of two phases: construction and local search. The construction phase

builds a feasible solution that is used as the starting solution for local search. The best solution over all GRASP iterations is returned as the result.

In the construction phase, a feasible solution is built, one element at a time. The set of candidate elements is made up of those elements that can be added to the current solution under construction without causing infeasibilities. A candidate element is evaluated by a greedy function that measures the local benefit of including that element in the partially constructed solution. The value-based restricted candidate list (RCL) is made up of candidate elements having a greedy function value at least as good as a specified threshold. The next element to be included in the solution is selected at random from the RCL. Its inclusion in the solution alters the greedy functions and the set of candidate elements used to determine the next RCL. The construction procedure terminates when the set of candidate elements is empty.

A local search algorithm successively replaces the current solution by a better solution in its neighborhood, if one exists. It terminates with a locally optimal solution when there is no better solution in the neighborhood. Since the solutions generated by a GRASP construction phase are usually sub-optimal, local search almost always improves the constructed solution.

GRASP has been used to find quality solutions for a wide range of combinatorial optimization problems (Festa and Resende, 2002). Furthermore, many extensions and improvements have been proposed for GRASP. Many of these extensions consist in the hybridization of the method with other methaheuristics. We observe that hybrid strategies usually find better solutions than those obtained using the pure GRASP, having in some cases improved the solution for open problems in the literature (Resende and Werneck, 2002; 2003; Ribeiro et al., 2002).

Hybrid strategies of GRASP with path-relinking have been leading to significant improvements in solution quality when compared to the solutions obtained by the pure method. Path-relinking was originally proposed by Glover (1996) as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search (Glover, 2000; Glover and Laguna, 1997; Glover et al., 2000). The use of path-relinking within a GRASP procedure, as an intensification strategy applied to each locally optimal solution, was first proposed by Laguna and Martí (1999). It was followed by several extensions, improvements, and successful applications (Aiex et al., 2003; 2000; Canuto et al., 2001; Resende and Ribeiro, 2003; Ribeiro et al., 2002).

The path-relinking approach consists in exploring trajectories that connect an *initial solution* and a *guiding solution*. This is done by introducing in the initial solution attributes of the guiding solution. At each step, all moves that incorporate attributes of the guiding solution are analyzed and the best move is chosen. In GRASP with path-relinking, one of these solutions is obtained by the local search phase of GRASP and the other is chosen among the solutions kept in an elite set of solutions found during the iterations of the algorithm.

Parallel computers have increasingly found their way into metaheuristics (Duni et al., 2002). A frequent question raised during the development of a parallel strategy is how the algorithm will behave when implemented in parallel as compared to a sequential implementation. The ideal situation is achieving linear speedup, i.e. when the

execution time of the sequential program divided by the execution time of the parallel program running in ρ processors is equal to ρ. Methodologies to help analyze the behavior of parallel strategies are significant in the development of a parallel application.

Most of the parallel implementations of GRASP found in the literature consist in either partitioning the search space or partitioning the GRASP iterations and assigning each partition to a processor. GRASP is applied to each partition in parallel. Examples of these strategies can be found in (Alvim and Ribeiro, 1998; Alvim, 1998; Feo et al., 1994; Li et al., 1994; Martins et al., 1999; 2000; 1998; Murphey et al., 1998; Pardalos et al., 1995; 1996; Resende et al., 1998).

For hybrid strategies of GRASP with path-relinking, two general parallelization approaches have been proposed. In the first one, named the *independent approach*, the communication among processors during GRASP iterations is limited to the detection of program termination. In the second approach, called the *cooperative approach*, processors share information on elite solutions visited during GRASP iterations. These strategies are classified according to Verhoeven and Aarts (1995) as *multiple independent trajectories* and *multiple cooperative trajectories*, respectively. Examples of parallel GRASP with path-relinking can be found in (Aiex et al., 2003; 2000; Canuto et al., 2001; Ribeiro and Rosseti, 2002).

In this paper, we analyze two parallel strategies for GRASP with path-relinking and propose a criterion to predict parallel efficiency based on experiments with a sequential implementation of the algorithm. Independent and cooperative parallel strategies are described and implemented for the 3-index assignment problem (AP3) and the job-shop scheduling problem (JSP).

The remainder of this paper is organized as follows. In Section 8.2, the sequential implementations of GRASP and of path-relinking are described for both the AP3 and the JSP. Section 8.3 shows how GRASP and path-relinking are combined. The parallel approaches are discussed in Section 8.4. The computational results are reported and analyzed in Section 8.5. Concluding remarks are made in Section 8.6.

8.2 SEQUENTIAL GRASP

In this section, the GRASP implementations developed for the 3-index assignment problem in Aiex et al. (2000) and for the job-shop scheduling problem in Aiex et al. (2003) are reviewed in Subsections 8.2.1 and 8.2.2, respectively. Path-relinking is generalized for these problems in Subsection 8.2.3.

8.2.1 GRASP for the AP3

The three-index assignment problem (AP3) was first stated by Pierskalla (1967) as a straightforward extension of the classical two-dimensional assignment problem. The AP3 is NP-Hard (Frieze, 1983; Garey and Johnson, 1979).

A formulation of the AP3 (often called the three-dimensional matching problem) is the following: Given three disjoint sets I, J, and K, such that $|I| = |J| = |K| = n$ and a weight c_{ijk} associated with each ordered triplet $(i, j, k) \in I \times J \times K$, find a minimum weight collection of n disjoint triplets $(i, j, k) \in I \times J \times K$.

The AP3 can also be formulated using permutation functions. There are n^3 cost elements and the optimal solution of the AP3 consists of the n smallest, such that the constraints are not violated. Assign to each set I, J, and K, the numbers $1,2,\ldots,n$. None of the chosen triplets (i,j,k) is allowed to have the same value for indices i, j, and k as another. For example, the choice of triplets $(1,2,4)$ and $(3,2,5)$ is infeasible, since these triplets share index $j=2$. The permutation-based formulation for the AP3 is

$$\min_{p,q\in\pi_N} \sum_{i=1}^{n} c_{ip(i)q(i)}$$

where π_N denotes the set of all permutations of the set of integers $N=\{1,2,\ldots,n\}$.

8.2.1.1 GRASP for the AP3 – Construction phase. The GRASP construction phase builds a feasible solution S by selecting n triplets, one at a time.

A restricted candidate list parameter α is selected at random from the interval $[0,1]$. This value is not changed during the construction phase. Solution S is initially empty and the set C of candidate triplets is initially the set of all triplets. To select the p-th ($1 \le p \le n-1$) triplet to be added to the solution, a restricted candidate list C' is defined to include all triplets (i,j,k) in the candidate set C having cost $c_{ijk} \le \underline{c} + \alpha(\overline{c} - \underline{c})$, where

$$\underline{c} = \min\{c_{ijk} \mid (i,j,k) \in C\} \text{ and } \overline{c} = \max\{c_{ijk} \mid (i,j,k) \in C\}.$$

Triplet $(i_p,j_p,k_p) \in C'$ is chosen at random and is added to the solution, i.e. $S = S \cup \{(i_p,j_p,k_p)\}$.

Once (i_p,j_p,k_p) is selected, the set of candidate triplets must be adjusted to take into account that (i_p,j_p,k_p) is part of the solution. Any triplet (i,j,k) such that $i=i_p$ or $j=j_p$ or $k=k_p$ must be removed from the current set of candidate triplets. This updating procedure is the computational bottleneck of the construction phase. A straightforward implementation would scan all n^3 cost elements $n-1$ times in order to update the candidate list. In Aiex et al. (2000), four doubly linked lists are used to implement this process more efficiently, reducing the complexity from $O(n^4)$ to $O(n^3)$.

After $n-1$ triplets have been selected, the set C of candidate triplets contains one last triplet which is added to S, thus completing the construction phase.

8.2.1.2 GRASP for the AP3 – Local search phase. The solution of the AP3 can be represented by a pair of permutations (p,q). Therefore, the solution space consists of all $(n!)^2$ possible combinations of permutations.

Let us first define the difference between two permutations s and s' to be

$$\delta(s,s') = \{i \mid s(i) \neq s'(i)\},$$

and the distance between them to be

$$d(s,s') = |\delta(s,s')|.$$

In this local search, a 2-exchange neighborhood is adopted. A 2-exchange neighborhood is defined to be

$$N_2(s) = \{s' \mid d(s,s') = 2\}.$$

The definition of the neighborhood $N(s)$ is crucial for the performance of the local search. In the 2-exchange neighborhood scheme used in this local search, the neighborhood of a solution (p,q) consists of all 2-exchange permutations of p plus all 2-exchange permutations of q. This means that for a solution $p,q \in \pi_N$, the 2-exchange neighborhood is

$$N_2(p,q) = \{p',q' \mid d(p,p') + d(q,q') = 2\}.$$

Hence, the size of the neighborhood is $|N_2(p)| + |N_2(q)| = 2\binom{n}{2}$. In the local search, the cost of each neighborhood solution is compared with the cost of the current solution. If the cost of the neighbor is lower, then the solution is updated, the search is halted, and a search in the new neighborhood is initialized. The local search ends when no neighbor of the current solution has a lower cost than the current solution.

8.2.2 GRASP for the JSP

The job shop scheduling problem (JSP) is a well-studied problem in combinatorial optimization. It consists in processing a finite set of jobs on a finite set of machines. Each job is required to complete a set of operations in a fixed order. Each operation is processed on a specific machine for a fixed duration. Each machine can process at most one job at a time and once a job initiates processing on a given machine it must complete processing on that machine without interruption. A schedule is a mapping of operations to time slots on the machines. The makespan is the maximum completion time of the jobs. The objective of the JSP is to find a schedule that minimizes the makespan. The JSP is NP-hard (Lenstra and Rinnooy Kan, 1979) and has also proven to be computationally challenging.

Mathematically, the JSP can be stated as follows. Given a set $\mathcal{M}$ of machines (where we denote the size of $\mathcal{M}$ by $|\mathcal{M}|$) and a set $\mathcal{J}$ of jobs (where the size of $\mathcal{J}$ is denoted by $|\mathcal{J}|$), let $\sigma_1^j \prec \sigma_2^j \prec \cdots \prec \sigma_{|\mathcal{M}|}^j$ be the ordered set of $|\mathcal{M}|$ operations of job j, where $\sigma_k^j \prec \sigma_{k+1}^j$ indicates that operation σ_{k+1}^j can only start processing after the completion of operation σ_k^j. Let O be the set of operations. Each operation σ_k^j is defined by two parameters: $\mathcal{M}_k^j$ is the machine on which σ_k^j is processed and $p_k^j = p(\sigma_k^j)$ is the processing time of operation σ_k^j. Defining $t(\sigma_k^j)$ to be the starting time of the k-th operation $\sigma_k^j \in O$, the JSP can be formulated as follows:

$$
\begin{aligned}
&\text{minimize } C_{\max} \\
&\text{subject to: } C_{\max} \geq t(\sigma_k^j) + p(\sigma_k^j), \text{ for all } \sigma_k^j \in O, \\
&\qquad t(\sigma_k^j) \geq t(\sigma_l^j) + p(\sigma_l^j), \text{ for all } \sigma_l^j \prec \sigma_k^j, && \text{(8.1a)} \\
&\qquad t(\sigma_k^j) \geq t(\sigma_l^i) + p(\sigma_l^i) \vee && \text{(8.1b)} \\
&\qquad t(\sigma_l^i) \geq t(\sigma_k^j) + p(\sigma_k^j), \text{ for all } \sigma_l^i, \sigma_k^j \in O \text{ such that } \mathcal{M}_{\sigma_l^i} = \mathcal{M}_{\sigma_k^j}, \\
&\qquad t(\sigma_k^j) \geq 0, \text{ for all } \sigma_k^j \in O,
\end{aligned}
$$

where C_{max} is the makespan to be minimized.

A feasible solution of the JSP can be built from a permutation of $\mathcal{J}$ on each of the machines in $\mathcal{M}$, observing the precedence constraints, the restriction that a machine can process only one operation at a time, and requiring that once started, processing of an operation must be uninterrupted until its completion. Since each set of feasible permutations has a corresponding schedule, the objective of the JSP is to find, among the feasible permutations, the one with the smallest makespan.

8.2.2.1 GRASP for the JSP – Construction phase. Consider the GRASP construction phase for the JSP, proposed in Binato et al. (2002) and Aiex et al. (2003), where a single operation is the building block of the construction phase. That is, a feasible schedule is built by scheduling individual operations, one at a time, until all operations have been scheduled.

While constructing a feasible schedule, not all operations can be selected at a given stage of the construction. An operation σ_k^j can only be scheduled if all prior operations of job j have already been scheduled. Therefore, at each construction phase iteration, at most $|\mathcal{J}|$ operations are candidates to be scheduled. Let this set of candidate operations be denoted by O_c and the set of already scheduled operations by O_s and denote the value of the greedy function for candidate operation σ_k^j by $h(\sigma_k^j)$.

The greedy choice is to next schedule operation $\underline{\sigma_k^j} = \text{argmin}(h(\sigma_k^j) \mid \sigma_k^j \in O_c)$. Let $\overline{\sigma_k^j} = \text{argmax}(h(\sigma_k^j) \mid \sigma_k^j \in O_c)$, $\underline{h} = h(\underline{\sigma_k^j})$, and $\overline{h} = h(\overline{\sigma_k^j})$. Then, the GRASP restricted candidate list (RCL) is defined as

$$\text{RCL} = \{\sigma_k^j \in O_c \mid \underline{h} \leq h(\sigma_k^j) \leq \underline{h} + \alpha(\overline{h} - \underline{h})\},$$

where α is a parameter such that $0 \leq \alpha \leq 1$.

A typical iteration of the GRASP construction is summarized as follows: a partial schedule (which is initially empty) is on hand, the next operation to be scheduled is selected from the RCL and is added to the partial schedule, resulting in a new partial schedule. The selected operation is inserted in the earliest available feasible time slot on machine $\mathcal{M}_{\sigma_k^j}$. Construction ends when the partial schedule is complete, i.e. all operations have been scheduled.

The algorithm uses two greedy functions. Even numbered iterations use a greedy function based on the makespan resulting from the inclusion of operation σ_k^j to the already-scheduled operations, i.e. $h(\sigma_k^j) = C_{max}$ for $O = \{O_s \cup \sigma_k^j\}$. In odd numbered

iterations, solutions are constructed by favoring operations from jobs having long remaining processing times. The greedy function used is given by $h(\sigma_k^j) = -\sum_{\sigma_l^j \notin O_s} p_l^j$, which measures the remaining processing time for job j. The use of two different greedy functions produce a greater diversity of initial solutions to be used by the local search.

8.2.2.2 GRASP for the JSP – Local search phase. To attempt to decrease the makespan of the solution produced in the construction phase, we employ the 2-exchange local search used in (Aiex et al., 2003; Binato et al., 2002; Taillard, 1991), that is based on the disjunctive graph model of Roy and Sussmann (1964).

The disjunctive graph $G = (V, A, E)$ is defined such that

$$V = \{O \cup \{0, |\mathcal{J}| \cdot |\mathcal{M}| + 1\}\}$$

is the set of nodes, where $\{0\}$ and $\{|\mathcal{J}| \cdot |\mathcal{M}| + 1\}$ are artificial source and sink nodes, respectively,

$$\begin{aligned} A = \{(v,w) \mid v,w \in O, v \prec w\} \cup \\ \{(0,w) \mid w \in O, \nexists v \in O \ni v \prec w\} \cup \\ \{(v,|\mathcal{J}| \cdot |\mathcal{M}| + 1) \mid v \in O, \nexists w \in O \ni v \prec w\} \end{aligned}$$

is the set of directed arcs connecting consecutive operations of the same job, and

$$E = \{(v,w) \mid \mathcal{M}_v = \mathcal{M}_w\}$$

is the set of edges that connect operations on the same machine. Vertices in the disjunctive graph model are weighted. Vertices 0 and $|\mathcal{J}| \cdot |\mathcal{M}| + 1$ have weight zero, while the weight of vertex $i \in \{1, \ldots, |\mathcal{J}| \cdot |\mathcal{M}|\}$ is the processing time of the operation corresponding to vertex i. Notice that the edges of A and E correspond, respectively, to constraints (8.1a) and (8.1b) of the disjunctive programming formulation of the JSP.

An orientation for the edges in E corresponds to a feasible schedule. Given an orientation of E, one can compute the earliest start time of each operation by computing the longest (weighted) path from node 0 to the node corresponding to the operation. Consequently, the makespan of the schedule can be computed by finding the critical (longest) path from node 0 to node $|\mathcal{J}| \cdot |\mathcal{M}| + 1$. The objective of the JSP is to find an orientation of E such that the longest path in G is minimized.

Taillard (1994) described an $O(|\mathcal{J}| \cdot |\mathcal{M}|)$ algorithm to compute the longest path on G. He also showed that the entire neighborhood of a given schedule, where the neighborhood is defined by the swap of two consecutive operations in the critical path, can be examined (i.e. have their makespan computed) in time $O(|\mathcal{J}| \cdot |\mathcal{M}|)$ given that the longest path of G was evaluated.

Given the schedule produced in the construction phase, the local search procedure initially identifies the critical path in the disjunctive graph corresponding to that schedule. All pairs of consecutive operations sharing the same machine in the critical path are tentatively exchanged. If the exchange improves the makespan, the move is accepted. Otherwise, the exchange is undone. Once an exchange is accepted, the critical

path may change and a new critical path must be identified. If no pairwise exchange of consecutive operations in the critical path improves the makespan, the current schedule is locally optimal and the local search ends.

8.2.3 Path-relinking

Using permutation arrays, we generalize path-relinking in this subsection for both the AP3 and the JSP.

A solution of the AP3 can be represented by two permutation arrays of numbers $1,2,\ldots,n$ in sets J and K, respectively, as follows:

$$S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,n}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,n})\},$$

where $j^S_{i,k}$ is the k-th number assigned to permutation i in solution S.

Analogously, for the JSP, a schedule can be represented by the permutation of operations in $\mathcal{J}$ on the machines in $\mathcal{M}$. The schedule is represented by $|\mathcal{M}|$ permutation arrays, each with $|\mathcal{J}|$ operations. Each permutation implies an ordering of the operations. A solution of the JSP is represented as follows:

$$S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,|\mathcal{J}|}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,|\mathcal{J}|}), \ldots, (j^S_{|\mathcal{M}|,1}, j^S_{|\mathcal{M}|,2}, \ldots, j^S_{|\mathcal{M}|,|\mathcal{J}|})\},$$

where $j^S_{i,k}$ is the k-th operation executed on machine i in solution S.

A path-relinking strategy for permutation arrays is carried out as follows. For a problem represented with R permutation arrays of elements from a set E, path-relinking is done between an initial solution

$$S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,|E|}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,|E|}), \ldots, (j^S_{R,1}, j^S_{R,2}, \ldots, j^S_{R,|E|})\}$$

and a guiding solution

$$T = \{(j^T_{1,1}, j^T_{1,2}, \ldots, j^T_{1,|E|}), (j^T_{2,1}, j^T_{2,2}, \ldots, j^T_{2,|E|}), \ldots, (j^T_{R,1}, j^T_{R,2}, \ldots, j^T_{R,|E|})\},$$

where $j^{S'}_{i,k}$ is the k-th element of permutation i in solution S'.

Let the difference between S and T be defined by the R sets of indices

$$\delta^{S,T}_k = \{i = 1, \ldots, |E| \mid j^S_{k,i} \neq j^T_{k,i}\}, k = 1, \ldots, R.$$

During a path-relinking move, a permutation array in S, given by

$$(\ldots, j^S_{k,i}, j^S_{k,i+1}, \ldots, j^S_{k,q-1}, j^S_{k,q}, \ldots),$$

is replaced by a permutation array

$$(\ldots, j^S_{k,q}, j^S_{k,i+1}, \ldots, j^S_{k,q-1}, j^S_{k,i}, \ldots),$$

by exchanging permutation elements $j^S_{k,i}$ and $j^S_{k,q}$, where $i \in \delta^{S,T}_k$ and q are such that $j^T_{k,q} = j^S_{k,i}$.

At each step of the algorithm, the move that produces the lowest cost solution is selected and its index is removed from the corresponding set $\delta^{S,T}_k$. This continues until there are only two move indices left in one of the sets $\delta^{S,T}_k$. At this point, the move obtained by exchanging these elements will produce the guiding solution. The best solution found during the path traversal is returned by the procedure.

8.3 GRASP WITH PATH-RELINKING

This section describes how path-relinking and GRASP can be combined to form a hybrid GRASP with path-relinking. We limit this discussion to single processor implementations and consider parallel strategies in the next section. Pseudo-code for the GRASP with path-relinking is presented in Figure 8.1. Let $|P|$ be the size of the current elite set and let *maxpool* be the the elite set's maximum size. The first *maxpool* GRASP iterations contribute one solution to the elite set per GRASP iteration (line 16). Path-relinking is not done until the pool of elite solutions is full.

In lines 3 and 4, GRASP construction and local search phases are carried out. Each of these phases can be replaced by more elaborated mechanisms, as for example, the construction phase developed for the JSP that alternates between two different greedy functions.

Once the pool of elite solutions is full, solution S produced by the local search phase of GRASP is tested to verify its quality (line 6). This is done to avoid relinking low-quality solutions. If S passes the quality test used, bidirectional path-relinking is done between S and all elements of a subset $P' \subseteq P$ (lines 7 to 15). In bidirectional path-relinking (Aiex et al., 2000), two paths are analyzed: one from the GRASP solution to the selected solution from the pool; another from the selected pool solution to the GRASP solution. The degree of restrictiveness of the quality test is a function of the computational time necessary to do path-relinking. For the AP3, the cost of a solution in the neighborhood of S can be computed from the cost of S in $O(1)$ time and therefore, path-relinking is applied to all solutions obtained in the local search phase of GRASP. For the JSP, on the other hand, the cost of each solution visited by path-relinking is computed in $O(|\mathcal{J}| \cdot |\mathcal{M}|)$ using the algorithm proposed in Taillard (1994). Therefore, it is computationally expensive to apply path-relinking after each GRASP iteration, and path-relinking is applied only when the GRASP solution satisfies a given quality criterion (Aiex et al., 2003). After each path-relinking phase, the best solution traversed during path-relinking is tested for inclusion in the elite pool (lines 11 and 13).

Every *ifreq* GRASP iterations, an intensification procedure is carried out (lines 17 to 19). The intensification procedure (Aiex et al., 2000) is accomplished by applying path-relinking to each pair of elite solutions in P and updating the pool when necessary. The procedure is repeated until no further change in P occurs.

The GRASP with path-relinking loop from line 2 to 24 continues for at most *maxitr* iterations, but can be terminated when a solution having a cost less than or equal to *look*4 is found (lines 21 to 23).

Finally, a path-relinking post optimization is done on the elite set (line 25). Path-relinking as a post-optimization step was introduced in Aiex et al. (2000) and Ribeiro et al. (2002) and has been also used in Resende and Werneck (2002; 2003). After applying path-relinking between all pairs of elite solutions without any change in the elite set, the local search procedure is applied to each elite solution, as the solutions produced by path-relinking are not always local optima. The local optima found are candidates for insertion into the elite set. If a change in the elite set occurs, the entire post-processing step is repeated.

```
procedure GRASP_PR(seed, look4, maxitr, maxpool, ifreq, problem_data)
1     P = ∅;
2     for i = 1, ..., maxitr do
3           CONSTRUCTION(seed, S, problem_data);
4           LOCAL(S, problem_data);
5           if |P| == maxpool then
6                 accepted = VERIFY_QUALITY(S);
7                 if accepted then
8                       select P' ⊆ P;
9                       for T ∈ P' do
10                            S_gmin = PATH_RELINKING(cost_S, S, T, problem_data);
11                            UPDATE_POOL(S_gmin, cost_gmin, P);
12                            S_gmin = PATH_RELINKING(cost_T, T, S, problem_data);
13                            UPDATE_POOL(S_gmin, cost_gmin, P);
14                      rof;
15                fi;
16          else P = P ∪ {S} fi;
17          if  mod (i,ifreq) == 0 then
18                INTENSIFY(P);
19          fi;
20          S_best = argmin(P);
21          if cost_best ≤ look4 then
22                break;
23          fi;
24    rof;
25    POSTOPT(P);
26    S_best = argmin(P);
27    return (S_best);
end GRASP_PR;
```

Figure 8.1 GRASP with path-relinking.

8.4 PARALLEL GRASP STRATEGIES

In this section, two parallel strategies for the GRASP with path-relinking algorithm shown in Figure 8.1 are described. The first scheme, called *independent*, limits communication between processors only to problem input, detection of process termination, and determination of best overall solution. In addition to the communication allowed in the independent scheme, the second scheme, called *cooperative*, allows processes to exchange information about their elite sets.

8.4.1 Independent parallel strategy

We revisit a basic parallelization scheme for GRASP with path-relinking used in Aiex et al. (2000). Figure 8.2 shows pseudo-code for this *multiple independent walks* scheme.

Our implementations use message passing for communication among processors. This communication is limited to program initialization and termination. When ρ processors are used, a single process reads the problem data and passes it to the remaining $\rho - 1$ processes. Processes send a message to all others when they either stop upon

finding a solution at least as good as the target value $look4$ or complete the maximum number of allotted iterations.

```
procedure INDEPENDENT_GRASP_PR(seed, look4, maxitr, maxpool, ifreq, problem_data)
1  my_rank = GET_RANK(); ρ = GET_NUM_PROCS();
2  for i = 1,...,(maxitr/ρ) * my_rank do
3      seed = rand(seed);
4  rof;
5  P = ∅; num_proc_stop = 0;
6  for i = 1,...,∞ do
7      CONSTRUCTION(seed, S, problem_data);
8      LOCAL(S, problem_data);
9      if |P| == maxpool then
10         accepted = VERIFY_QUALITY(S);
11         if accepted then
12             select P' ⊆ P;
13             for T ∈ P' do
14                 S_gmin = PATH_RELINKING(cost_S, S, T, problem_data);
15                 UPDATE_POOL(S_gmin, cost_gmin, P);
16                 S_gmin = PATH_RELINKING(cost_T, T, S, problem_data);
17                 UPDATE_POOL(S_gmin, cost_gmin, P);
18             rof;
19         fi;
20     else P = P ∪ {S} fi;
21     if  mod (i,ifreq) == 0 then INTENSIFY(P); fi;
22     S_best = argmin(P);
23     if cost_best ≤ look4 then SEND_ALL(look4_stop) fi;
24     if i == maxitr/ρ then
25         num_proc_stop = num_proc_stop + 1;
26         SEND_ALL(maxitr_stop);
27     fi;
28     received = VERIFY_RECEIVING(flag);
29     if received then
30         if flag == look4_stop then break;
31         else if flag == maxitr_stop then
32             num_proc_stop = num_proc_stop + 1;
33         fi;
34     fi;
35     if num_proc_stop == ρ then break fi;
36 rof;
37 POSTOPT(P);
38 S_GlobalBest = GET_GLOBAL_BEST(S_best);
39 return (S_GlobalBest);
end INDEPENDENT_GRASP_PR;
```

Figure 8.2 Pseudo-code for the independent parallel GRASP with path-relinking.

The independent parallel GRASP with path-relinking is built upon the sequential algorithm of Figure 8.1. Each process executes a copy of the program. We discuss the differences between the sequential algorithm and this parallel variant. In line 1 of Figure 8.2, the rank *my_rank* of the process and the number ρ of processes are determined. Each GRASP construction phase is initialized with a random number generator seed. To assure independence of processes, identical seeds of the random

number generator (`rand()`) must not be used by more than one process to initiate a construction phase. The initial seed for process *my_rank* is computed in lines 2 to 4. We attempt, this way, to increase the likelihood that each process has a disjunct sequence of $maxitr/\rho$ initial seeds.

The **for** loop from line 6 to line 36 executes the iterations. The construction, local search, and path-relinking phases are identical to the those of the sequential algorithm. In line 23, if a process finds a solution with cost less than or equal to $look4$, it sends a flag to each of the other processes indicating that it has found the solution. Likewise, when a process completes $maxitr/\rho$ iterations, it sends a different flag to each of the other processes indicating that it has completed the preset number of iterations (lines 24 to 27). In line 28, the process checks if there are any status flags to be received. If there is a flag indicating that a solution with cost not greater than $look4$ has been found, the iterations are terminated in line 30. If a flag indicating that some process has completed the preset number of iterations has been received, then a counter num_proc_stop of the number of processes that are ready to be terminated is incremented (line 32). If all processes have completed their iterations, the execution of the main **for** loop is terminated (line 35).

Each process, upon terminating the **for** loop going from line 6 to line 36, runs the post-optimization phase on the pool of elite solutions (line 37). A reduce operator (`GET_GLOBAL_BEST`) determines the global best solution among all processes in line 38.

A pure GRASP parallel algorithm can be obtained from the algorithm in Figure 8.2 by skipping the execution of lines 9 to 19. As in a basic GRASP, it is necessary to keep track of the best solution found and no pool handling operations are necessary. Therefore, intensification and post-optimization are not defined in a parallel implementation of pure GRASP.

8.4.2 Cooperative parallel strategy

In the cooperative parallel GRASP with path-relinking, processes share elite set information. We now describe this scheme, whose pseudo-code is presented in Figure 8.3. This algorithm is built on top of the independent scheme presented in the previous subsection. We limit our discussion to the differences between the two schemes, which occur in the path-relinking phase.

Before doing path-relinking between solutions S and T, each process checks if one or more other processes have sent it new elite solutions. If there are new elite solutions to be received, `RECEIVE_SOLUTIONS` (in lines 14 and 18) receives the elite solutions, tests if each elite solution can be accepted for insertion into its local elite set, and inserts any accepted elite solution. Upon termination of each path-relinking leg, if the local elite set is updated, then (in lines 17 and 21) the process writes the new elite solutions to a local send buffer. In line 23, if the local send buffer is not empty, the process sends the buffer's contents to the other processes.

Another difference between the independent and the cooperative schemes concerns the `INTENSIFY` procedure. In the cooperative scheme, whenever the local elite set pool is updated, the new elite solutions are written to the send buffer. These bufferized solutions will be sent to the other processes the next time that procedure `SEND_SOLUTIONS` is invoked.

8.5 COMPUTATIONAL RESULTS

This section reports on computational experiments with the parallel versions of the pure GRASP and GRASP with path-relinking proposed in Section 8.4. The parallel strategies have been implemented for both the AP3 and the JSP described in Section 8.2.

8.5.1 Computer environment

The experiments were done on an SGI Challenge computer (16 196-MHz MIPS R10000 processors and 12 194-MHz MIPS R10000 processors) with 7.6 Gb of memory. The algorithms were coded in Fortran and were compiled with the SGI MIPSpro F77 compiler using flags `-O3 -static -u`. The parallel codes used SGI's Message Passing Toolkit 1.4, which contains a fully compliant implementation of version 1.2 of the Message-Passing Interface (MPI) (Snir et al., 1998) specification. In the parallel experiments, wall clock times were measured with the MPI function `MPI_WT`. This is also the case for runs with a single processor that are compared to multiple-processor runs. Timing in the parallel runs excludes the time to read the problem data, to initialize the random number generator seeds, and to output the solution.

The parallel implementations were run on 2, 4, 8, and 16 processors. Load on the machine was low throughout the experiments, therefore processors were always available. The average speedups were computed dividing the sum of the execution times of the independent parallel program executing on one processor by the sum of the execution times of the parallel program on 2, 4, 8, and 16 processors, for 60 runs. The execution times of the independent parallel program executing on one processor and the execution times of the sequential program are approximately the same.

8.5.2 The parallel GRASP strategies

The following parallel algorithms were studied in these experiments:

1. pure GRASP;
2. independent GRASP with path-relinking;
3. cooperative GRASP with path-relinking.

Path-relinking was always applied between the GRASP solution and all solutions in the elite set.

The parallel GRASP as well as the parallel GRASP with path-relinking used in the experiments are named `GRASP`$(prob)$ and `GRASP` $+$ `PR`$(prob)$, where *prob* indicates the problem type (AP3 or JSP). The parameters of the procedures used in the parallel approaches of GRASP for the AP3 and for the JSP are the same used for testing the sequential algorithms in Aiex et al. (2000) and Aiex et al. (2003), respectively. Intensification and post-optimization are not carried out during the experiments with the parallel implementations.

8.5.3 Test problems

For the AP3, we tested one problem of each size $n = 20, 22, 24, 26$, generated by Balas and Saltzman (1991). We named these problems `B-S 20.1`, `B-S 22.1`, `B-S 24.1` and `B-S 26.1`. For the JSP we tested problems `abz6`, `mt10`, `orb5`, and `la21` from four classes of standard problems for the JSP. These problems were obtained from Beasley's OR-Library [1] (Beasley, 1990).

8.5.4 Probability distribution of solution time to target value

Aiex et al. (2002) studied the empirical probability distributions of the random variable *time to target value* in GRASP. They showed that, given a target solution value, the time GRASP takes to find a solution with cost at least as good as the target fits a two-parameter exponential distribution. Empirical distributions are produced from experimental data and corresponding theoretical distributions are estimated from the empirical distributions.

A quantile-quantile plot (Q-Q plot) and a plot showing the empirical and the theoretical distributions of the random variable time to target value for sequential GRASP and GRASP with path-relinking for the AP3 are shown in Figures 8.4 and 8.5, respectively. Analogously, Figures 8.6 and 8.7 show the same plots for the JSP. These plots are computed by running the algorithms for 200 independent runs. Each run ends when the algorithm finds a solution with value less than or equal to a specified target value (*look4*). Each running time is recorded and the times are sorted in increasing order. We associate with the i-th sorted running time (t_i) a probability $p_i = (i - \frac{1}{2})/200$, and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 200$ as the empirical distribution.

Following Chambers et al. (1983), we determine the theoretical quantile-quantile plot for the data to estimate the parameters of the two-parameter exponential distribution. To describe Q-Q plots, recall that the cumulative distribution function for the two-parameter exponential distribution is given by

$$F(t) = 1 - e^{-(t-\mu)/\lambda},$$

where λ is the mean and standard deviation of the distribution data and μ is the shift of the distribution with respect to the ordinate axis. For each value p_i, $i = 1, \ldots, 200$, we associate a p_i-quantile $Qt(p_i)$ of the theoretical distribution. For each p_i-quantile we have, by definition, that

$$F((Qt(p_i)) = p_i.$$

Hence, $Qt(p_i) = F^{-1}(p_i)$ and therefore, for the two-parameter exponential distribution, we have

$$Qt(p_i) = -\lambda \ln(1 - p_i) + \mu.$$

The quantiles of the data of an empirical distribution are simply the (sorted) raw data.

A theoretical quantile-quantile plot (or theoretical Q-Q plot) is obtained by plotting the quantiles of the data of an empirical distribution against the quantiles of a theoretical distribution. This involves three steps. First, the data (in our case, the measured

[1] `http://mscmga.ms.ic.ac.uk/jeb/orlib/jobshopinfo.html`

times) are sorted in ascending order. Second, the quantiles of the theoretical exponential distribution are obtained. Finally, a plot of the data against the theoretical quantiles is made.

When the theoretical distribution is a close approximation of the empirical distribution, the points in the Q-Q plot will have a nearly straight configuration. If the parameters λ and μ of the theoretical distribution that best fits the measured data could be estimated a priori, the points in a Q-Q plot would tend to follow the line $x = y$. Alternatively, in a plot of the data against a two-parameter exponential distribution with $\lambda' = 1$ and $\mu' = 0$, the points would tend to follow the line $y = \lambda x + \mu$. Consequently, parameters λ and μ of the two-parameter exponential distribution can be estimated, respectively, by the slope and the intercept of the line depicted in the Q-Q plot.

To avoid possible distortions caused by outliers, we do not estimate the distribution mean by linear regression on the points of the Q-Q plot. Instead, we estimate the slope $\hat{\lambda}$ of line $y = \lambda x + \mu$ using the upper quartile q_u and lower quartile q_l of the data. The upper and lower quartiles are, respectively, the $Q(\frac{1}{4})$ and $Q(\frac{3}{4})$ quantiles, respectively. We take

$$\hat{\lambda} = (z_u - z_l)/(q_u - q_l)$$

as an estimate of the slope, where z_u and z_l are the u-th and l-th points of the ordered measured times, respectively. These estimates are used to plot the theoretical distributions on the plots on the right side of the figures.

The lines above and below the estimated line on the Q-Q plots correspond to plus and minus one standard deviation in the vertical direction from the line fitted to the plot. This superimposed variability information is used to analyze the straightness of the Q-Q plots.

The following can be stated for a two parameter (shifted) exponential distribution (Aiex et al., 2002; Verhoeven and Aarts, 1995). Let $P_\rho(t)$ be the probability of not having found a given (target) solution in t time units with ρ independent processes. If $P_1(t) = e^{-(t-\mu)/\lambda}$ with $\lambda \in \mathbb{R}^+$ and $\mu \in \mathbb{R}$, i.e. P_1 corresponds to a two parameter exponential distribution, then $P_\rho(t) = e^{-\rho(t-\mu)/\lambda}$. This follows from the definition of the two-parameter exponential distribution. It implies that the probability of finding a solution of a given value in time ρt with a sequential process is equal to $1 - e^{-(\rho t-\mu)/\lambda}$ while the probability of finding a solution at least as good as that given value in time t with ρ independent parallel processes is $1 - e^{-\rho(t-\mu)/\lambda}$. Note that if $\mu = 0$, then both probabilities are equal and correspond to the non-shifted exponential distribution. Furthermore, since $\rho \geq 1$, if $\rho|\mu| \ll \lambda$, then the two probabilities are approximately equal and it is possible to approximately achieve linear speed-up in solution time to target value using multiple independent processes.

8.5.5 A test to predict speedup of parallel implementations

The observation above suggests a test using a sequential implementation to determine whether it is likely that a parallel implementation using multiple independent processors will be efficient. We say a parallel implementation is efficient if it achieves linear speedup (with respect to wall time) to find a solution at least as good as a given target value (*look4*). The test consists in K (200, for example) independent runs of the sequential program to build a Q-Q plot and estimate the parameters μ and λ of the shifted

Table 8.1 Speedup and efficiency for instances of the AP3. Algorithm is the parallel implementation of GRASP. Instances are B-S 20.1, B-S 22.1, B-S 24.1, and B-S 26.1, with target values 16, 16, 16, and 17, respectively. The estimated parameters for the exponential distributions are shown for each pair of instance/target value.

	estimated parameter			number of processors							
				2		4		8		16	
prob.	μ	λ	$\lvert\mu\rvert/\lambda$	spdup	effic.	spdup	effic.	spdup	effic.	spdup	effic.
B-S 20.1	1.28	90.18	.014	2.02	1.01	3.66	.91	6.05	.75	16.30	1.01
B-S 22.1	-2.607	185.21	.014	2.03	1.01	4.58	1.14	10.33	1.29	17.88	1.11
B-S 24.1	-2.890	246.55	.011	2.16	1.08	4.27	1.06	7.89	.98	13.91	.86
B-S 26.1	26.36	252.90	.104	1.62	.81	3.22	.80	6.23	.77	11.72	.73
average:			**.034**	**1.95**	**.97**	**3.93**	**.97**	**7.62**	**.95**	**14.95**	**.93**

Table 8.2 Speedups for instances of the AP3. Algorithms are independent and cooperative implementations of GRASP with path-relinking. Instances are B-S 20.1, B-S 22.1, B-S 24.1, and B-S 26.1, with target values 7, 8, 7, and 8, respectively.

	estimated parameter			speedup independent (number of processors)				speedup cooperative (number of processors)			
prob.	μ	λ	$\lvert\mu\rvert/\lambda$	2	4	8	16	2	4	8	16
B-S 20.1	-26.46	1223.80	.021	1.67	3.34	6.22	10.82	1.56	3.47	7.37	14.36
B-S 22.1	-135.12	3085.32	.043	2.25	4.57	9.01	14.37	1.64	4.22	8.83	18.78
B-S 24.1	-16.76	4004.11	.004	1.71	4.00	7.87	12.19	2.16	4.00	9.38	19.29
B-S 26.1	32.12	2255.55	.014	2.11	3.89	6.10	11.49	2.16	5.30	9.55	16.00
average:			**.020**	**1.935**	**3.95**	**7.3**	**12.21**	**1.88**	**4.24**	**8.78**	**17.10**

exponential distribution. If $\rho|\mu| \ll \lambda$, then we predict that the parallel implementation will be efficient.

8.5.6 The parallel experiments

The goals of the experiments in this subsection are threefold. First, we attempt to verify computationally the validity of the test to predict speedup of parallel implementations proposed above. Second, we contrast independent parallel implementations of pure GRASP with GRASP with path-relinking. Finally, we compare independent parallel implementations of GRASP with path-relinking with cooperative implementations. Because of the nature of these experiments, the stopping criterion for maximum number of iterations was disabled, and the programs terminated only when a solution with cost as good as the target value was found.

In Aiex et al. (2000) and Aiex et al. (2003), we verified that the times to target solution in the GRASP variants for the AP3 and the JSP fit a two-parameter exponential distribution.

8.5.6.1 Parallel results for the AP3. To study the parallel implementation of GRASP(*AP3*), we tested problems B-S 20.1, B-S 22.1, B-S 24.1, and B-S 26.1 with target values 16, 16, 16, and 17, respectively. The independent and cooperative

parallel implementations of GRASP + PR(*AP3*) were studied for problems B-S 20.1, B-S 22.1, B-S 24.1, and B-S 26.1, with target values 7, 8, 7, and 8, respectively.

The speedups and efficiencies (speedup divided by the number of processors used) observed for GRASP(*AP3*) are shown in Table 8.1. We also show the estimates for parameters μ and λ of the two-parameter exponential distribution, as well as the value of $|\mu|/\lambda$ for each pair of instance/target value tested. By examining the parameters estimated for GRASP(*AP3*), we can group the instances into two categories: { B-S 20.1, B-S 22.1, B-S 24.1 } and { B-S 26.1 }. For instances B-S 20.1, B-S 22.1, and B-S 24.1, the ratio $|\mu|/\lambda$ is approximately 0.01, while for runs on problem B-S 26.1, it is about 10 times greater. By our proposed criterion, we would expect that the runs in the first category present better speedups than the one in the other category, which indeed is what one observes. The super-linear speedups observed in problem B-S 22.1 are probably explained by statistical fluctuation. Some negative estimates for μ observed in the table should be expected whenever the shift in the exponential distribution is small, since μ is the intercept of the ordinate axis of the line defined by the first and third quartiles of the Q-Q plot.

The speedups for the independent and cooperative parallel approaches of GRASP + PR(*AP3*) are shown in Figure 8.8. In Table 8.2, the speedups observed in the plots of Figure 8.8 are summarized. The estimated values of μ, λ, and $|\mu|/\lambda$, using 200 independent runs of the sequential GRASP + PR(*AP3*) are also shown. An approximately linear speedup is observed for the independent GRASP + PR(*AP3*) for up to 8 processors. With 16 processors, we observe a reduction in the speedup, although we can still consider that the program scales well for up to 16 processors. The gradual degradation in speedup as the number of processors increases is expected and is due to the fact that the number of processors ρ offsets the ratio $|\mu|/\lambda$, i.e. $|\mu|/\lambda \leq 2|\mu|/\lambda \leq 4|\mu|/\lambda \leq 8|\mu|/\lambda \leq 16|\mu|/\lambda$.

For the cooperative parallel GRASP + PR(*AP3*), we cannot make any prediction based on the ratio $|\mu|/\lambda$, since the processors share information and are therefore not independent. We observe that the cooperative approach benefited more than the independent approach from the increase in the number of processors. The increase in sharing of elite solutions compensated for the increase in inter-process communication. In fact, super-linear speedups can occur for the cooperative approach.

8.5.6.2 Parallel results for the JSP. The parallel GRASP(*JSP*) was tested on instances abz6, mt10, orb5, and la21, with target values 960, 960, 920, and 1120, respectively. The independent and cooperative parallel GRASP + PR(*JSP*), were also tested on instances abz6, mt10, orb5, and la21, but with more difficult target values 943, 938, 895, and 1100, respectively.

Table 8.3 lists the values of μ, λ, and $|\mu|/\lambda$ for each tested pair of instance and target value, as well as the speedups and efficiencies for the 2, 4, 8, and 16-processor runs. Speedups are, on average, approximately linear, in accordance with the low values observed for $|\mu|/\lambda$ in the table.

The plots in Figure 8.9 show speedup for both parallel implementations of GRASP + PR(*JSP*). Table 8.4 summarizes the speedups in Figure 8.9. The values of μ, λ, and $|\mu|/\lambda$ are also shown. In accordance with the speedup prediction test, sub-linear speedups are observed for the independent approach of GRASP + PR(*JSP*). We notice that the ratios

Table 8.3 Speedup and efficiency for instances of the JSP. Algorithm is the parallel implementation of GRASP. Instances are abz6, mt10, orb5, and la21, with target values 960, 960, 920, and 1120, respectively. The estimated parameters for the exponential distributions are shown for each pair of instance/target value.

	estimated parameter			number of processors							
				2		4		8		16	
prob.	μ	λ	$\lvert\mu\rvert/\lambda$	spdup	effic.	spdup	effic.	spdup	effic.	spdup	effic.
abz6	.42	15.56	.027	2.04	1.02	4.75	1.18	8.87	1.10	19.17	1.19
mt10	11.72	885.03	013	1.62	.81	4.07	1.01	7.34	.91	14.81	.92
orb5	1.24	38.27	.032	2.12	1.06	3.97	.99	7.63	.95	14.10	.88
la21	-1.01	206.83	.005	1.94	.97	4.98	1.24	8.13	1.01	19.63	1.22
average:			**.019**	**1.93**	**.96**	**4.44**	**1.10**	**7.99**	**.99**	**16.92**	**1.05**

Table 8.4 Speedups for instances of the JSP. Algorithms are independent, and cooperative implementation of GRASP with path-relinking. Instances are abz6, mt10, orb5, and la21, with target values 943, 938, 895, and 1100, respectively. The estimated parameters for the exponential distributions are shown for each pair of instance/target value.

	estimated parameter			speedup independent (number of processors)				speedup cooperative (number of processors)			
prob.	μ	λ	$\lvert\mu\rvert/\lambda$	2	4	8	16	2	4	8	16
abz6	47.67	756.56	.06	2.00	3.36	6.44	10.51	2.40	4.21	11.43	23.58
mt10	305.27	524.23	.58	1.57	2.12	3.03	4.05	1.75	4.58	8.36	16.97
orb5	130.12	395.41	.32	1.95	2.97	3.99	5.36	2.10	4.91	8.89	15.76
la21	175.20	407.73	.42	1.64	2.25	3.14	3.72	2.23	4.47	7.54	11.41
average:			**.34**	**1.79**	**2.67**	**4.15**	**5.91**	**2.12**	**4.54**	**9.05**	**16.93**

$|\mu|/\lambda$ are much higher than those in Table 8.3 for GRASP(JSP), as well as for ratios computed for the AP3 instances. On the other hand, the table shows linear and superlinear speedups for most of the instances tested with the cooperative approach. These speedups are considerably higher than those observed for the independent approach. For example, for 16 processors, the average speedups for the cooperative approach are almost three times higher than those of the independent approach. These results show that cooperation is more critical in the JSP than in the AP3. This perhaps is because path-relinking is applied less frequently in the JSP runs than in the AP3 runs, due to the criterion used to avoid applying path-relinking to poor quality solutions in the JSP.

Figure 8.10 shows the empirical distributions for the parallel GRASP(JSP), and the parallel independent and cooperative GRASP + PR(JSP), using 16 processors. The programs were tested for problems abz6, mt10, orb5 and la21, with target values 943, 938, 895, and 1100, respectively. We notice that for 16 processors, although parallel GRASP(JSP) scales better than the independent GRASP + PR(JSP), the execution times of the latter are in general, lower than the execution times of the former. We also notice that the execution times of the cooperative GRASP + PR(JSP) are considerably lower than the execution times of the parallel GRASP(JSP) and of the independent GRASP + PR(JSP).

8.6 CONCLUSION

Parallel computers have increasingly found their way into metaheuristics. In particular, many parallel implementations of GRASP have been described in the literature. Recently, parallel implementations of GRASP with path-relinking have been proposed. In this paper, we contrast independent and cooperative parallel schemes for GRASP with path-relinking with a parallel scheme for GRASP. We also propose a test using a sequential GRASP implementation to predict the speedup of independent parallel implementations.

Implementations for the 3-index assignment problem and the job-shop scheduling problem are described. Computational experiments are done on four instances from each problem.

We conclude that the proposed test is useful for predicting the degree of speedup expected for parallel implementations of GRASP and GRASP with path-relinking. The results also show that parallel GRASP with path-relinking outperforms parallel GRASP and that the cooperative scheme outperforms the independent approach, often achieving super-linear speedups.

```
procedure COOPERATIVE_GRASP_PR(seed, look4, maxitr, maxpool, ifreq, problem_data)
1   my_rank = GET_RANK(); ρ = GET_NUM_PROCS();
2   for i = 1,...,(maxitr/ρ) * my_rank do
3       seed = rand(seed);
4   rof;
5   P = ∅; num_proc_stop = 0;
6   for i = 1,...,∞ do
7       CONSTRUCTION(seed, S, problem_data);
8       LOCAL(S, problem_data);
9       if |P| == maxpool then
10          accepted = VERIFY_QUALITY(S);
11          if accepted then
12              select P' ⊆ P;
13              for T ∈ P' do
14                  RECEIVE_SOLUTIONS(P);
15                  S_gmin = PATH_RELINKING(cost_S, S, T, problem_data);
16                  updated = UPDATE_POOL(S_gmin, cost_gmin, P);
17                  if (updated) then INSERT_SEND_BUFFER(S_gmin, cost_gmin,buffer) fi;
18                  RECEIVE_SOLUTIONS(P);
19                  S_gmin = PATH_RELINKING(cost_T, T, S, problem_data);
20                  updated = UPDATE_POOL(S_gmin, cost_gmin, P);
21                  if (updated) then INSERT_SEND_BUFFER(S_gmin, cost_gmin,buffer) fi;
22              rof;
23              SEND_SOLUTIONS(buffer);
24          fi;
25      else P = P ∪ {S} fi;
26      if  mod (i,ifreq) == 0 then INTENSIFY(P) fi;
27      S_best = argmin(P);
28      if cost_best ≤ look4 then SEND_ALL(look4_stop) fi;
29      if i == maxitr/ρ then
30          num_proc_stop = num_proc_stop + 1;
31          SEND_ALL(maxitr_stop)
32      fi;
33      received = VERIFY_RECEIVING(flag);
34      if received then
35          if flag == look4_stop then break;
36          else if flag == maxitr_stop then
37              num_proc_stop = num_proc_stop + 1;
38          fi;
39      fi;
40      if num_proc_stop == ρ then break fi;
41  rof;
42  POSTOPT(P);
43  S_GlobalBest = GET_GLOBAL_BEST(S_best);
44  return (S_GlobalBest);
end COOPERATIVE_GRASP_PR;
```

Figure 8.3 Pseudo-code for the cooperative parallel GRASP with path-relinking.

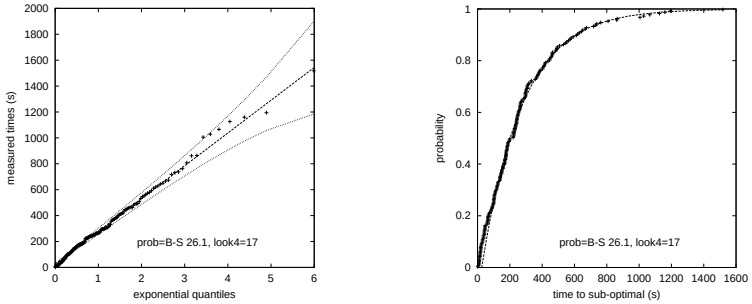


Figure 8.4 Exponential distribution and Q-Q plot for GRASP for the AP3: problem `B-S 26.1` with target value (*look4*) of 17.

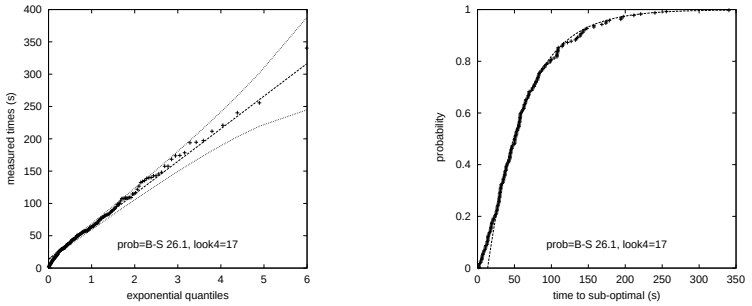


Figure 8.5 Exponential distribution and Q-Q plot for GRASP with path-relinking for the AP3: problem `B-S 26.1` with target value (*look4*) of 17.

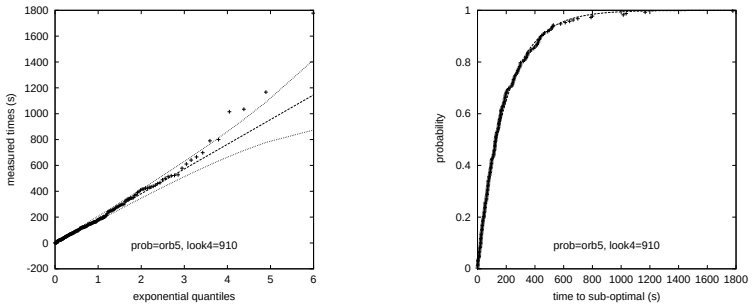


Figure 8.6 Exponential distribution and Q-Q plot for GRASP for the JSP: problem `orb5` with target value (*look4*) of 910.

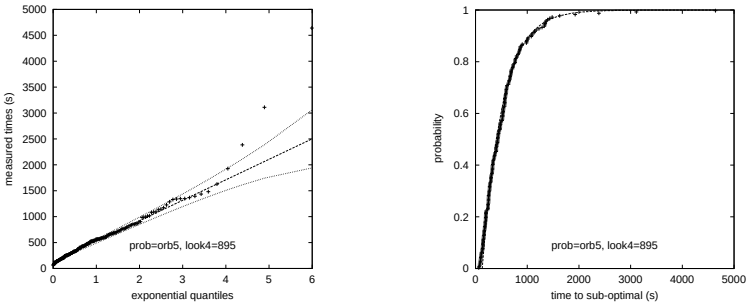


Figure 8.7 Exponential distribution and Q-Q plot for GRASP with path-relinking for the JSP: problem `orb5` with target value (*look4*) of 895.

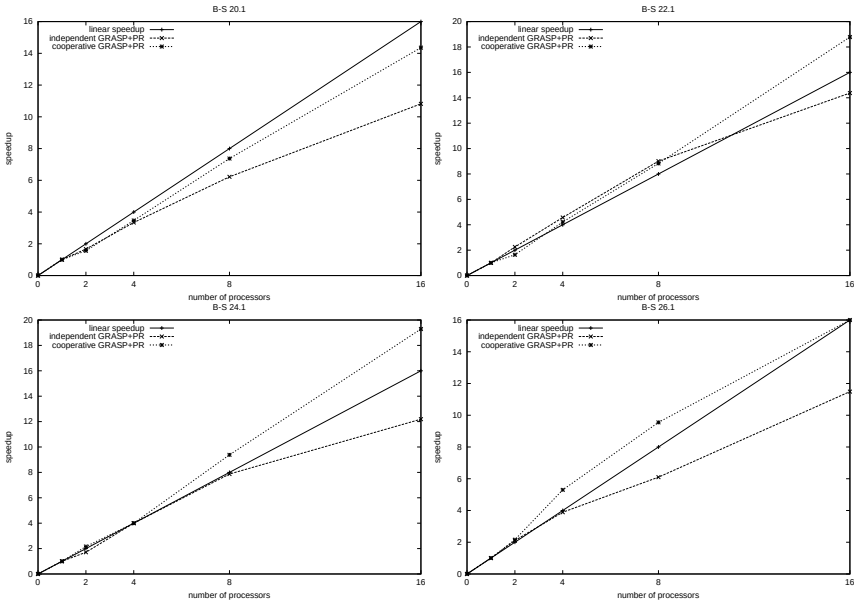


Figure 8.8 Speedups for the parallel implementations of independent and cooperative GRASP with path-relinking for the AP3: problems `B-S 20.1`, `B-S 22.1`, `B-S 24.1`, and `B-S 26.1` with target values 7, 8, 7, and 8, respectively.

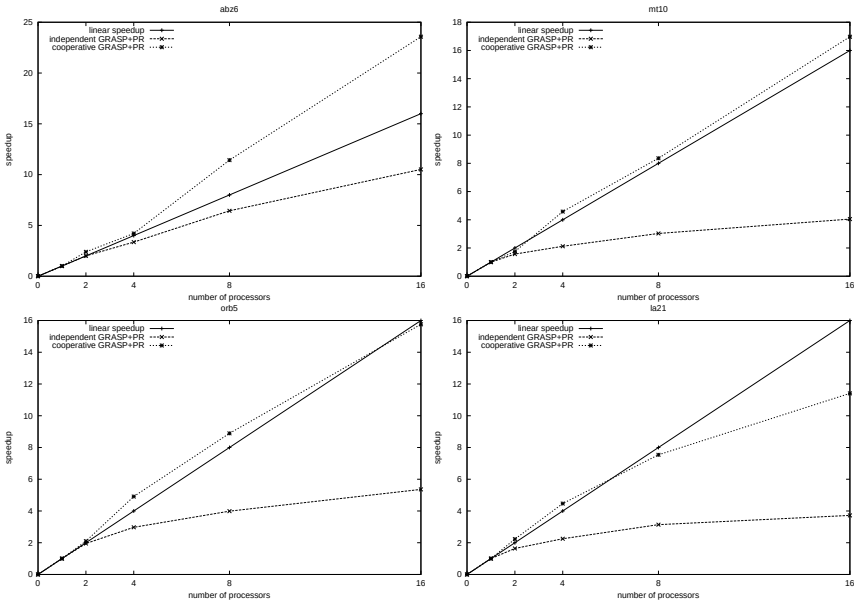


Figure 8.9 Speedups for the parallel implementations of independent and cooperative GRASP with path-relinking for the JSP: problems `abz6`, `mt10`, `orb5`, and `la21` with target values 943, 938, 895, and 1100, respectively.

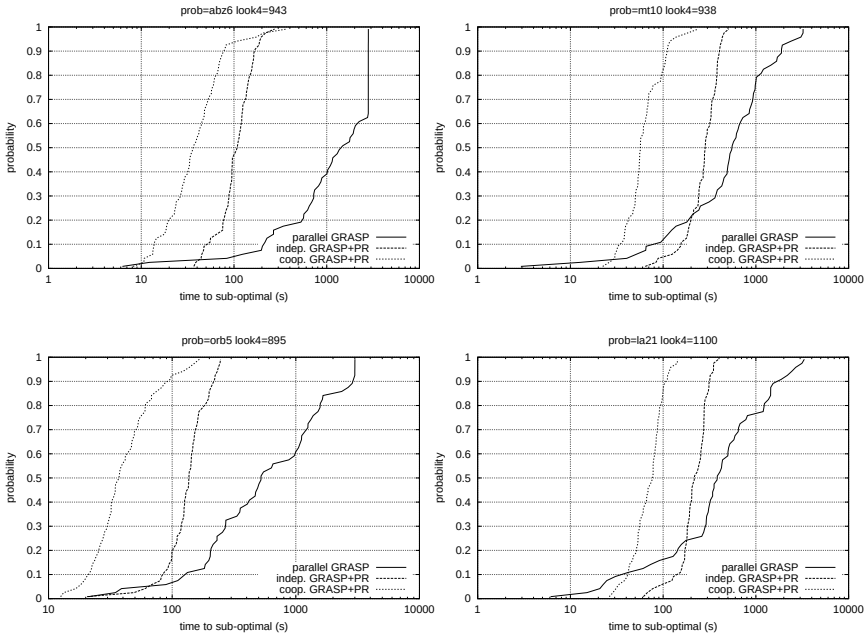


Figure 8.10 Empirical distributions for 16 processor parallel implementations of pure GRASP, independent GRASP with path-relinking, and cooperative GRASP with path-relinking: problems `abz6`, `mt10`, `orb5` and `la21`, for target values 943, 938, 895, and 1100, respectively.

Bibliography

R.M. Aiex, S. Binato, and M.G.C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29:393–430, 2003.

R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with path relinking for the three-index assignment problem. Technical report, Information Sciences Research Center, AT&T Labs Research, Florham Park, NJ 07932 USA, 2000. To appear in *INFORMS Journal on Computing*.

R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8:343–373, 2002.

A. Alvim and C.C. Ribeiro. Balanceamento de carga na paralelização da metaheurística GRASP. In *X Simpósio Brasileiro de Arquiteturas de Computadores*, pages 279–282. Sociedade Brasileira de Computação, 1998.

A.C.F. Alvim. Estratégias de paralelização da metaheurística GRASP. Master's thesis, Departamento de Informática, PUC-Rio, Rio de Janeiro, RJ 22453-900 Brazil, April 1998.

E. Balas and M.J. Saltzman. An algorithm for the three-index assignment problem. *Oper. Res.*, 39:150–161, 1991.

J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

S. Binato, W.J. Hery, D.M. Loewenstern, and M.G.C. Resende. A GRASP for job shop scheduling. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys on Metaheuristics*, pages 58–79. Kluwer Academic Publishers, 2002.

S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38:50–58, 2001.

J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. *Graphical Methods for Data Analysis*. Chapman & Hall, 1983.

S. Duni, P.M. Pardalos, and M.G.C. Resende. Parallel metaheuristics for combinatorial optimization. In R. Corrêa, I. Dutra, M. Fiallos, and F. Gomes, editors, *Models for Parallel and Distributed Computation – Theory, Algorithmic Techniques and Applications*, pages 179–206. Kluwer Academic Publishers, 2002.

T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.

P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

A.M. Frieze. Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research*, 13:161–164, 1983.

M.R. Garey and D.S. Johnson. *Computers and intractability - A guide to the theory of NP-completeness*. W.H. Freeman and Company, 1979.

F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

F. Glover. Multi-start and strategic oscillation methods – Principles to exploit adaptive memory. In M. Laguna and J.L. Gonzáles-Velarde, editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pages 1–24. Kluwer, 2000.

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. Technical report, Graduate School of Business and Administration, University of Colorado, Boulder, CO 80309-0419, 2000.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

J. K. Lenstra and A. H. G. Rinnooy Kan. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.

Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994.

S.L. Martins, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Greedy randomized adaptive search procedures for the Steiner problem in graphs. In P.M. Pardalos, S. Rajasejaran, and J. Rolim, editors, *Randomization methods in algorithmic design*, volume 43 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 133–145. American Mathematical Society, 1999.

S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P.M. Pardalos. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization*, pages 267–283, 2000.

S.L. Martins, C.C. Ribeiro, and M.C. Souza. A parallel GRASP for the Steiner problem in graphs. In A. Ferreira and J. Rolim, editors, *Proceedings of IRREGUALAR'98 – 5th International Symposium on Solving Irregularly Structured Problems in Parallel*, volume 1457 of *Lecture Notes in Computer Science*, pages 285–297. Springer-Verlag, 1998.

R.A. Murphey, P.M. Pardalos, and L.S. Pitsoulis. A parallel GRASP for the data association multidimensional assignment problem. In P.M. Pardalos, editor, *Parallel processing of discrete problems*, volume 106 of *The IMA Volumes in Mathematics and Its Applications*, pages 159–180. Springer-Verlag, 1998.

P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP implementation for the quadratic assignment problem. In A. Ferreira and J. Rolim, editors, *Parallel Algorithms for Irregularly Structured Problems – Irregular'94*, pages 115–133. Kluwer Academic Publishers, 1995.

P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP for MAX-SAT problems. *Lecture Notes in Computer Science*, 1184:575–585, 1996.

W.P. Pierskalla. The tri-subsitution method for the three-multidimensional assignment problem. *CORS Journal*, 5:71–81, 1967.

M.G.C. Resende, T.A. Feo, and S.H. Smith. Algorithm 787: Fortran subroutines for approximate solution of maximum independent set problems using GRASP. *ACM Trans. Math. Software*, 24:386–394, 1998.

M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2002.

M.G.C. Resende and C.C. Ribeiro. A GRASP with path-relinking for private virtual circuit routing. *Networks*, 41:104–114, 2003.

M.G.C. Resende and R.F. Werneck. A hybrid heuristic for the p-median problem. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ, 2002.

M.G.C. Resende and R.F. Werneck. A hybrid multistart heuristic for the uncapacitated facility location problem. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ, 2003.

C.C. Ribeiro and I. Rosseti. A parallel GRASP for the 2-path network design problem. *Lecture Notes in Computer Science*, 2004:922–926, 2002.

C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14:228–246, 2002.

B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives, 1964.

M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI – The complete reference, Volume 1 – The MPI Core*. The MIT Press, 1998.

E. D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6:108–117, 1994.

E.D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.

M.G.A. Verhoeven and E.H.L. Aarts. Parallel local search. *Journal of Heuristics*, 1: 43–66, 1995.

9 PARALLEL GRASP WITH PATH-RELINKING FOR JOB SHOP SCHEDULING

Renata M. Aiex[1], Silvio Binato[2], and Mauricio G. C. Resende[3]

[1]Department of Computer Science
Catholic University of Rio de Janeiro
Rio de Janeiro, RJ 22453-900 Brazil
rma@inf.puc-rio.br

[2]Electrical Energy Research Center (CEPEL)
Rio de Janeiro, RJ 21944-970 Brazil
silvio@cepel.br

[3]Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA.
mgcr@research.att.com

Abstract: In the job shop scheduling problem (JSP), a finite set of jobs is processed on a finite set of machines. Each job is required to complete a set of operations in a fixed order. Each operation is processed on a specific machine for a fixed duration. A machine can process no more than one job at a time and once a job initiates processing on a given machine it must complete processing without interruption. A schedule is an assignment of operations to time slots on the machines. The objective of the JSP is to find a schedule that minimizes the maximum completion time, or makespan, of the jobs. In this paper, we describe a parallel greedy randomized adaptive search procedure (GRASP) with path-relinking for the JSP. A GRASP is a metaheuristic for combinatorial optimization. It usually consists of a construction procedure based on a greedy randomized algorithm and of a local search. Path-relinking is an intensification strategy that explores trajectories that connect high quality solutions. Independent and cooperative parallelization strategies are described and implemented. Computational experience on a large set of standard

test problems indicates that the parallel GRASP with path-relinking finds good-quality approximate solutions of the job shop scheduling problem.
Keywords: Combinatorial optimization, job shop scheduling, local search, GRASP, path-relinking, probabilistic algorithm.

9.1 INTRODUCTION

The job shop scheduling problem (JSP) is a well-studied problem in combinatorial optimization. It consists in processing a finite set of jobs on a finite set of machines. Each job is required to complete a set of operations in a fixed order. Each operation is processed on a specific machine for a fixed duration. Each machine can process at most one job at a time and once a job initiates processing on a given machine it must complete processing on that machine without interruption. A schedule is a mapping of operations to time slots on the machines. The makespan is the maximum completion time of the jobs. The objective of the JSP is to find a schedule that minimizes the makespan.

Mathematically, the JSP can be stated as follows. Given a set $\mathcal{M}$ of machines (where we denote the size of $\mathcal{M}$ by $|\mathcal{M}|$) and a set $\mathcal{J}$ of jobs (where the size of $\mathcal{J}$ is denoted by $|\mathcal{J}|$), let $\sigma_1^j \prec \sigma_2^j \prec \cdots \prec \sigma_{|\mathcal{M}|}^j$ be the ordered set of $|\mathcal{M}|$ operations of job j, where $\sigma_k^j \prec \sigma_{k+1}^j$ indicates that operation σ_{k+1}^j can only start processing after the completion of operation σ_k^j. Let O be the set of operations. Each operation σ_k^j is defined by two parameters: $\mathcal{M}_k^j$ is the machine on which σ_k^j is processed and $p_k^j = p(\sigma_k^j)$ is the processing time of operation σ_k^j. Defining $t(\sigma_k^j)$ to be the starting time of the k-th operation $\sigma_k^j \in O$, the JSP can be formulated as follows:

$$
\begin{aligned}
&\text{minimize } C_{\max} \\
&\text{subject to: } C_{\max} \geq t(\sigma_k^j) + p(\sigma_k^j), \text{ for all } \sigma_k^j \in O, \\
&\qquad t(\sigma_k^j) \geq t(\sigma_l^j) + p(\sigma_l^j), \text{ for all } \sigma_l^j \prec \sigma_k^j, && \text{(9.1a)} \\
&\qquad t(\sigma_k^j) \geq t(\sigma_l^i) + p(\sigma_l^i) \vee && \text{(9.1b)} \\
&\qquad t(\sigma_l^i) \geq t(\sigma_k^j) + p(\sigma_k^j), \text{ for all } \sigma_l^i, \sigma_k^j \in O \text{ such that } \mathcal{M}_{\sigma_l^i} = \mathcal{M}_{\sigma_k^j}, \\
&\qquad t(\sigma_k^j) \geq 0, \text{ for all } \sigma_k^j \in O,
\end{aligned}
$$

where C_{max} is the makespan to be minimized.

A feasible solution of the JSP can be built from a permutation of $\mathcal{J}$ on each of the machines in $\mathcal{M}$, observing the precedence constraints, the restriction that a machine can process only one operation at a time, and requiring that once started, processing of an operation must be uninterrupted until its completion. Once the permutation of $\mathcal{J}$ is given, its feasibility status can be determined in $O(|\mathcal{J}| \cdot |\mathcal{M}|)$ time. The feasibility-checking procedure determines the makespan $\mathcal{C}_{max}$ for feasible schedules (Taillard, 1994). Since, each set of feasible permutations has a corresponding schedule, the objective of the JSP is to find, among the feasible permutations, the one with the smallest makespan.

The JSP is NP-hard (Lenstra and Rinnooy Kan, 1979) and has also proven to be computationally challenging. Exact methods (Applegate and W.Cook, 1991; Brucker et al., 1994; Carlier and Pinson, 1989; 1990; Giffler and Thompson, 1960) have been successful in solving small instances, including the notorious 10×10 instance of (Fisher and Thompson, 1963), proposed in 1963 and only solved twenty years later. Problems of dimension 15×15 are still considered to be beyond the reach of today's exact methods. For such problems there is a need for good heuristics. Surveys of heuristic methods for the JSP are given in (Pinson, 1995; Vaessens et al., 1996). These include dispatching rules reviewed in French (1982), the shifting bottleneck approach (Adams et al., 1988; Applegate and W.Cook, 1991), local search (Lourenço, 1995; Lourenço and Zwijnenburg, 1996; Vaessens et al., 1996), simulated annealing (Van Laarhoven et al., 1992; Lourenço, 1995), tabu search (Taillard, 1994; Nowicki and Smutnicki, 1996; Lourenço and Zwijnenburg, 1996), and genetic algorithms (Davis, 1985). Recently, Binato et al. (2001) described a greedy randomized adaptive search procedure (GRASP) for the JSP. A comprehensive survey of job shop scheduling techniques can be found in Jain and Meeran (1998). In this paper, we present a new parallel GRASP with path-relinking for the job shop scheduling problem.

The remainder of the paper is organized as follows. In Section 9.2, we describe the new GRASP, describing two construction mechanisms and a local search algorithm. Path-relinking for the JSP and its incorporation to a GRASP are described in Section 9.3. Two parallelization schemes are presented in Section 9.4. Computational results are reported in Section 9.5 and concluding remarks are made in Section 9.6.

9.2 GRASP FOR JSP

GRASP (Feo and Resende, 1989; 1995; Festa and Resende, 2001; Resende and Ribeiro, 2001b) is an iterative process, where each GRASP iteration usually consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is explored by local search. The best solution over all GRASP iterations is returned as the result.

In the construction phase, a feasible solution is built, one element at a time. The set of candidate elements is made up of those elements that can be added to the current solution under construction without causing infeasibilities. A candidate element is evaluated by a greedy function which measures the local benefit of including that element in the constructed solution. The restricted candidate list (RCL) is made up of candidate elements with a greedy function value above a specified threshold. The next element to be included in the solution is selected at random from the RCL. Its inclusion in the solution alters the greedy functions and the set of candidate elements used to determine the next RCL. The construction procedure terminates when the set of candidate elements is empty.

Since the solutions generated by a GRASP construction phase are not guaranteed to be locally optimal, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm successively replaces the current solution by a better solution from its neighborhood. It terminates when there is no better solution found in the neighborhood.

In the remainder of this section, we describe two construction procedures and a commonly used local search strategy.

9.2.1 Construction procedures

For the JSP, we consider a single operation to be the building block of the construction phase. That is, we build a feasible schedule by scheduling individual operations, one at a time, until all operations are scheduled.

Recall that σ_k^j denotes the k-th operation of job j and is defined by the pair $(\mathcal{M}_k^j, p_k^j)$, where $\mathcal{M}_k^j$ is the machine on which operation σ_k^j is performed and p_k^j is the processing time of operation σ_k^j. While constructing a feasible schedule, not all operations can be selected at a given stage of the construction. An operation σ_k^j can only be scheduled if all prior operations of job j have already been scheduled. Therefore, at each construction phase iteration, at most $|\mathcal{J}|$ operations are candidates to be scheduled. Let this set of candidate operations be denoted by O_c and the set of already scheduled operations by O_s and denote the value of the greedy function for candidate operation σ_k^j by $h(\sigma_k^j)$.

The greedy choice is to next schedule operation $\underline{\sigma_k^j} = \text{argmin}(h(\sigma_k^j) \mid \sigma_k^j \in O_c)$. Let $\overline{\sigma_k^j} = \text{argmax}(h(\sigma_k^j) \mid \sigma_k^j \in O_c)$, $\underline{h} = h(\underline{\sigma_k^j})$, and $\overline{h} = h(\overline{\sigma_k^j})$. Then, the GRASP restricted candidate list (RCL) is defined as

$$\text{RCL} = \{\sigma_k^j \in O_c \mid \underline{h} \leq h(\sigma_k^j) \leq \underline{h} + \alpha(\overline{h} - \underline{h})\},$$

where α is a parameter such that $0 \leq \alpha \leq 1$.

A typical iteration of the GRASP construction is summarized as follows: a partial schedule (which is initially empty) is on hand, the next operation to be scheduled is selected from the RCL and is added to the partial schedule, resulting in a new partial schedule. The selected operation is inserted in the earliest available feasible time slot on machine $\mathcal{M}_{\sigma_k^j}$. Let a and b denote the start and end times of an available time slot on $\mathcal{M}_{\sigma_k^j}$ and let $e = t(\sigma_{k-1}^j) + p_{k-1}^j$ denote the completion time of operation σ_{k-1}^j. Insertion in this time slot at time point $\max\{a, e\}$ is feasible if and only if $b - \max\{a, e\} \geq p_k^j$. Construction ends when the partial schedule is complete, i.e. all operations have been scheduled.

In Binato et al. (2001), the greedy function $h(\sigma_k^j)$ is the makespan resulting from the inclusion of operation σ_k^j to the already-scheduled operations, i.e. $h(\sigma_k^j) = C_{max}$ for $O = \{O_s \cup \sigma_k^j\}$. We will refer to this greedy function as the *makespan* greedy function.

In this paper, we propose another greedy function, which, as we will see later, is used in conjunction with the makespan greedy function. This function favors operations from jobs having long remaining processing times by using the greedy function $h(\sigma_k^j) = -\sum_{\sigma_l^j \notin O_s} p_l^j$, which measures the remaining processing time for job j. We refer to it as the *time-remaining* greedy function.

9.2.2 *Local search phase*

Since there is no guarantee that the schedule obtained in the construction phase is optimal, local search should be applied to attempt to decrease its makespan.

We employ the two exchange local search, based on the disjunctive graph model of Roy and Sussmann (1964), and used in Binato et al. (2001). The disjunctive graph $G = (V, A, E)$ is defined such that

$$V = \{O \cup \{0, |\mathcal{J}| \cdot |\mathcal{M}| + 1\}\}$$

is the set of nodes, where $\{0\}$ and $\{|\mathcal{J}| \cdot |\mathcal{M}| + 1\}$ are artificial source and sink nodes, respectively,

$$\begin{aligned} A = \{(v,w) \mid v,w \in O, v \prec w\} \cup \\ \{(0,w) \mid w \in O, \nexists v \in O \ni v \prec w\} \cup \\ \{(v, |\mathcal{J}| \cdot |\mathcal{M}| + 1) \mid v \in O, \nexists w \in O \ni v \prec w\} \end{aligned}$$

is the set of directed arcs connecting consecutive operations of the same job, and

$$E = \{(v,w) \mid \mathcal{M}_v = \mathcal{M}_w\}$$

is the set of edges that connect operations on the same machine. Vertices in the disjunctive graph model are weighted. Vertices 0 and $|\mathcal{J}| \cdot |\mathcal{M}| + 1$ have weight zero, while the weight of vertex $i \in \{1, \ldots, |\mathcal{J}| \cdot |\mathcal{M}|\}$ is the processing time of the operation corresponding to vertex i. Notice that the edges of A and E correspond, respectively, to constraints (9.1a) and (9.1b) of the disjunctive programming formulation of the JSP.

An orientation for the edges in E corresponds to a feasible schedule. Given an orientation of E, one can compute the earliest start time of each operation by computing the longest (weighted) path from node 0 to the node corresponding to the operation. Consequently, the makespan of the schedule can be computed by finding the critical (longest) path from node 0 to node $|\mathcal{J}| \cdot |\mathcal{M}| + 1$. Thus, the objective of the JSP is to find an orientation of E such that the longest path in G is minimized.

Taillard (1994) describes an $O(|\mathcal{J}| \cdot |\mathcal{M}|)$ algorithm to compute the longest path on G and an $O(|\mathcal{J}| \cdot |\mathcal{M}|)$ procedure to recompute the makespan when two consecutive operations on the same machine in the critical path (on the same machine) are swapped. He also shows that the entire neighborhood of a given schedule, where the neighborhood is defined by the swap of two consecutive operations in the critical path, can be examined, i.e. have their makespan computed, in complexity $O(|\mathcal{J} \cdot \mathcal{M}|)$ given that the longest path of G was evaluated. These procedures were implemented in Binato et al. (2001) and are borrowed in our implementation.

Given the schedule produced in the construction phase, the local search procedure initially identifies the critical path in the disjunctive graph corresponding to that schedule. All pairs of consecutive operations sharing the same machine in the critical path are tentatively exchanged. If the exchange improves the makespan, the move is accepted. Otherwise, the exchange is undone. Once an exchange is accepted, the critical path may change and a new critical path must be identified. If no pairwise exchange of consecutive operations in the critical path improves the makespan, the current schedule is locally optimal and the local search ends.

9.3 PATH-RELINKING FOR JSP

```
procedure PATH_RELINKING (M, J, O, p, Makespan, S, T)
1  S = {(j^S_{1,1}, j^S_{1,2}, ..., j^S_{1,|J|}), (j^S_{2,1}, j^S_{2,2}, ..., j^S_{2,|J|}), ...,
                                                  (j^S_{|M|,1}, j^S_{|M|,2}, ..., j^S_{|M|,|J|})};
2  T = {(j^T_{1,1}, j^T_{1,2}, ..., j^T_{1,|J|}), (j^T_{2,1}, j^T_{2,2}, ..., j^T_{2,|J|}), ...,
                                                  (j^T_{|M|,1}, j^T_{|M|,2}, ..., j^T_{|M|,|J|})};
3  c_gmin = Makespan; S_gmin = S;
4  for k = 1, ..., |M| do
5      δ^{S,T}_k = {i = 1, ..., |J| | j^S_{k,i} ≠ j^T_{k,i}};
6  od;
7  while (Σ_{k=1}^{|M|} |δ^{S,T}_k| > 2) do
8      c_min = ∞;
9      for k = 1, ..., |M| do
10         for i ∈ δ^{S,T}_k do
11             Let q be such that j^T_{k,q} == j^S_{k,i};
12             S̄ = S \ {(..., j^S_{k,i}, j^S_{k,i+1}, ..., j^S_{k,q-1}, j^S_{k,q}, ...)};
13             S̄ = S̄ ∪ {(..., j^S_{k,q}, j^S_{k,i+1}, ..., j^S_{k,q-1}, j^S_{k,i}, ...)};
14             c̄ = CALCULATE_MAKESPAN (S̄);
15             if c̄ ≤ c_min then
16                 c_min = c̄;
17                 S_min = S̄;
18                 i_min = i;
19                 k_min = k;
20             fi;
21         rof;
22     rof;
23     S = S_min; Makespan = c_min;
24     δ^{S,T}_{k_min} = δ^{S,T}_{k_min} \ {i_min};
25     if Makespan ≤ c_gmin then
26         c_gmin = Makespan;
27         S_gmin = S;
28     fi;
29 elihw;
30 return (S_gmin);
end PATH_RELINKING;
```

Figure 9.1 Path-relinking between initial solution S and guiding solution T.

Path-relinking is an enhancement to the basic GRASP procedure, leading to significant improvements in solution quality. Path-relinking was originally proposed by Glover (1996) as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search (Glover, 2000; Glover and Laguna, 1997; Glover et al., 2000). The use of path-relinking within a GRASP procedure, as an intensification strategy applied to each locally optimal solution, was first proposed by Laguna and Martí (1999). It was followed by several extensions, improvements, and successful applications (Aiex et al., 2000a; Canuto et al., 2001; Resende and Ribeiro, 2001a; Ribeiro et al., 2001).

In this section, we propose a path-relinking strategy for the JSP. In our description, a schedule is represented by the permutation of operations in $\mathcal{J}$ on the machines in $\mathcal{M}$.

```
procedure GRASP_PR (M, J, O, p, look4, maxitr, maxpool, freq)
1  P = ∅;
2  for i = 1, ... , maxitr do
3      if  mod (i, 2) == 0 then
4          GREEDY_MASKESPAN(S, M, p, |M|, |J|, Makespan);
5      else
6          GREEDY_TIME_REMAINING(S, M, p, |M|, |J|, Makespan);
7      fi;
8      LOCAL(S, M, p, |M|, |J|, Makespan);
9      if |P| == maxpool then
10         accepted = VERIFY_QUALITY(S, i);
11         if accepted then
12             for T ∈ P' ⊆ P do
13                 S_gmin = PATH_RELINKING(M, J, O, p, Makespan, S, T);
14                 UPDATE_POOL(S_gmin, c_gmin, P);
15                 S_gmin = PATH_RELINKING(M, J, O, p, Makespan, T, S);
16                 UPDATE_POOL(S_gmin, c_gmin, P);
17             rof;
18         fi;
19     else P = P ∪ {S} fi;
20     if  mod (i, ifreq) == 0 then INTENSIFY(P) fi;
21     S_best = POOLMIN(P);
22     if MAKESPAN(S_best) ≤ look4 then return (S_best) fi;
23 rof;
24 POST_OPTIMIZATION(P);
25 S_best = POOLMIN(P);
26 return (S_best);
end GRASP_PR;
```

Figure 9.2 GRASP with bidirectional path-relinking for JSP.

The schedule is represented by $|\mathcal{M}|$ permutation arrays, each with $|\mathcal{J}|$ operations. Each permutation implies an ordering of the operations. A solution of the JSP is represented as follows:

$$S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,|\mathcal{J}|}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,|\mathcal{J}|}), \ldots, (j^S_{|\mathcal{M}|,1}, j^S_{|\mathcal{M}|,2}, \ldots, j^S_{|\mathcal{M}|,|\mathcal{J}|})\},$$

where $j^S_{i,k}$ is the k-th operation executed on machine i in solution S.

The path-relinking approach consists in exploring trajectories that connect a *initial solution* and a *guiding solution*. This is done by introducing in the initial solution attributes of the guiding solution. At each step, all moves that incorporate attributes of the guiding solution are analyzed and the best move is chosen. Usually the guiding solution is of high quality. For the JSP, path-relinking is done between an initial solution

$$S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,|\mathcal{J}|}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,|\mathcal{J}|}), \ldots, (j^S_{|\mathcal{M}|,1}, j^S_{|\mathcal{M}|,2}, \ldots, j^S_{|\mathcal{M}|,|\mathcal{J}|})\}$$

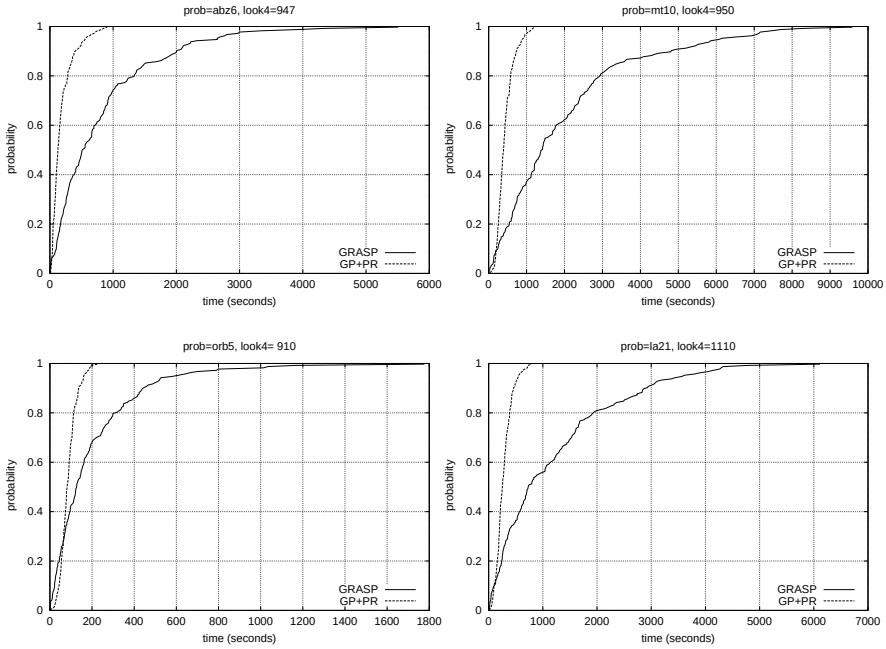


Figure 9.3 Empirical probability distributions of time to target value for GRASP and GP+PR: problems abz6, mt10, orb5 and la21.

and a guiding solution

$$T = \{(j^T_{1,1}, j^T_{1,2}, \ldots, j^T_{1,|\mathcal{J}|}), (j^T_{2,1}, j^T_{2,2}, \ldots, j^T_{2,|\mathcal{J}|}), \ldots, (j^T_{|\mathcal{M}|,1}, j^T_{|\mathcal{M}|,2}, \ldots, j^T_{|\mathcal{M}|,|\mathcal{J}|})\}.$$

Pseudo-code for this procedure is shown in Figure 9.1.

Let the symmetric difference between S and T be defined by the $|\mathcal{M}|$ sets of indices

$$\delta^{S,T}_k = \{i = 1, \ldots, |\mathcal{J}| \mid j^S_{k,i} \neq j^T_{k,i}\}, k = 1, \ldots, |\mathcal{M}|.$$

These sets are computed in lines 4 to 6 in the pseudo-code.

An intermediate solution of the path-relinking trajectory is visited at each step of the loop from line 7 to 29. During a move, a permutation array in S, given by

$$(\ldots, j^S_{k,i}, j^S_{k,i+1}, \ldots, j^S_{k,q-1}, j^S_{k,q}, \ldots),$$

is replaced by a permutation array

$$(\ldots, j^S_{k,q}, j^S_{k,i+1}, \ldots, j^S_{k,q-1}, j^S_{k,i}, \ldots),$$

by exchanging operations $j^S_{k,i}$ and $j^S_{k,q}$, where $i \in \delta^{S,T}_k$ and q are such that $j^T_{k,q} = j^S_{k,i}$. Note that solutions that violate the precedence constraints can be produced by these moves. The feasibility of solution S is verified during procedure *Calculate_Makespan*(S)

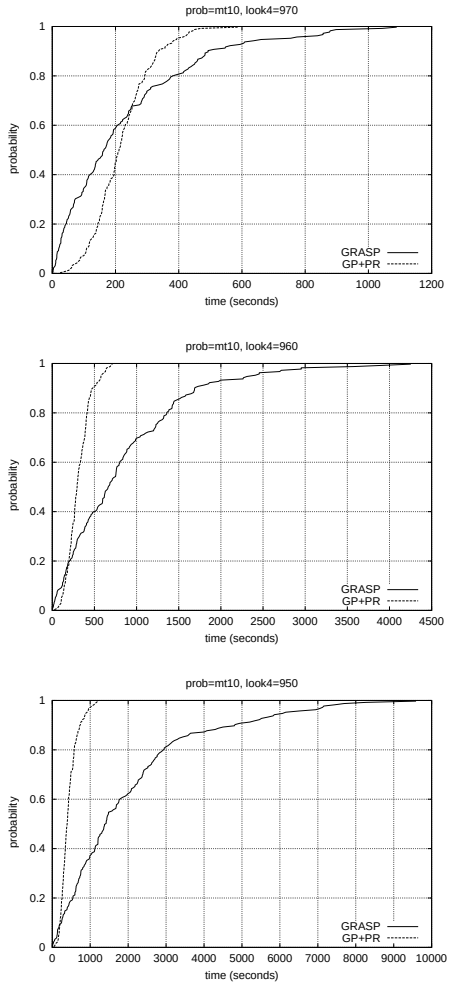


Figure 9.4 Empirical probability distributions of time to target value for GRASP and GP+PR: problem mt10 and target values 970, 960 and 950.

(line 14), which consists in computing the critical path in the disjunctive graph presented in Section 9.2.2, using the algorithm proposed in Taillard (1994). An infeasible schedule is detected when a cycle is found in the corresponding graph. The makespans of infeasible solutions are defined to be infinite so as to avoid visiting them in a path-relinking trajectory.

At each step of the algorithm, the move that produces the lowest cost solution is selected and its index is removed from the corresponding set $\delta_k^{S,T}$ (line 24). This continues until there are only two move indices left in one of the sets $\delta_k^{S,T}$. At this point,

pargjss-

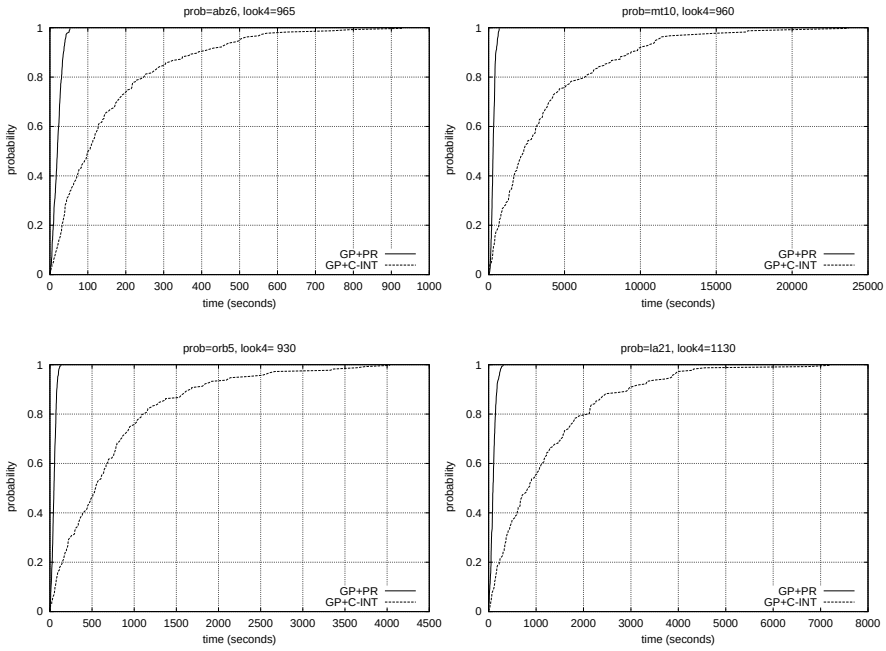


Figure 9.5 Empirical probability distributions of time to target value for `GP+C-INT` and `GP+PR`: problems `abz6`, `mt10`, `orb5` and `la21`.

the move obtained by exchanging these elements will produce the guiding solution. The best solution (S_{gmin}) found during the path traversal is returned by the procedure.

In the implementation proposed for the JSP, a *pool P* of elite solutions is built with the GRASP solutions produced during the first $|P|$ GRASP iterations. After this initial phase, a solution s_g produced by GRASP is relinked with one or more elite solutions s_e in P. Path-relinking can be applied from GRASP solution s_g to pool solution s_e, from pool solution s_e to GRASP solution s_g, or in both directions. These two trajectories very often visit different intermediate solutions.

The hybrid strategy proposed uses an approach developed by Fleurent and Glover (1999) to incorporate elite solutions to a GRASP. Let c_{best} and c_{worst} be the values of the objective functions of the best and the worst solution in P, respectively. Given two solutions

$$S = \{(j^S_{1,1}, j^S_{1,2}, \ldots, j^S_{1,|\mathcal{J}|}), (j^S_{2,1}, j^S_{2,2}, \ldots, j^S_{2,|\mathcal{J}|}), \ldots, (j^S_{|\mathcal{M}|,1}, j^S_{|\mathcal{M}|,2}, \ldots, j^S_{|\mathcal{M}|,|\mathcal{J}|})\}$$

and

$$T = \{(j^T_{1,1}, j^T_{1,2}, \ldots, j^T_{1,|\mathcal{J}|}), (j^T_{2,1}, j^T_{2,2}, \ldots, j^T_{2,|\mathcal{J}|}), \ldots, (j^T_{|\mathcal{M}|,1}, j^T_{|\mathcal{M}|,2}, \ldots, j^T_{|\mathcal{M}|,|\mathcal{J}|})\},$$

let

$$\Delta(S,T) = \sum_{k=1}^{|\mathcal{M}|} |\delta_k^{S,T}|$$

be a measure of the non-similarity between S and T.

A solution S_{gmin} produced by path-relinking is a candidate for insertion in the pool. S_{gmin} will be accepted if it satisfies one of the following acceptance criteria:

1. $c_{gmin} < c_{best}$, i.e., S_{gmin} is the best solution found so far;
2. $c_{best} \leq c_{gmin} < c_{worst}$ and for all elite solutions $S_p \in P$, $\Delta(S_{gmin}, S_p) > \Delta_{\min}$, i.e., S_{gmin} is better then the worst solution in P and differs significantly from all elite solutions.

Once accepted for insertion in P, S_{gmin} will replace the worst elite solution, which will be discarded from P.

Note that if the number of moves needed to traverse the path from $S \in P$ to S_{gmin} is at most $\Delta_{\min}/2$, then path solution S_{gmin} must satisfy $\Delta(S, S_{gmin}) \leq \Delta_{\min}$ and there is no need to compute the symmetric difference since S_{gmin} can only be included in the elite set if it is better than the best elite solution.

In the GRASP with path-relinking for the 3-index assignment problem (Aiex et al., 2000a), the cost of a solution in the neighborhood of S can be computed from the cost of S in O(1) time. For the JSP, the cost of each solution visited by path-relinking is computed in $O(|\mathcal{J}| \cdot |\mathcal{M}|)$, using the algorithm proposed in Taillard (1994). Therefore, it is computationally expensive to apply path-relinking after each iteration of the GRASP. Instead, we propose to apply path-relinking only when the GRASP solution satisfies a given quality criteria.

The quality criteria proposed uses the mean value μ_n and the standard deviation σ_n of the costs of the GRASP solutions produced during the first n iterations. A solution S_i takes part in path-relinking if

1. $c(S_i) \leq c_{worst}$, for $i \leq n$;
2. $c(S_i) \leq \max(c_{worst}, \mu_n - 2 * \sigma_n)$, for $i > n$.

Path-relinking can also be used in an intensification scheme for the elite set (Aiex et al., 2000a). This is accomplished by applying path-relinking to each pair of elite solutions in P and updating the pool when necessary. The procedure is repeated until no further change in P occurs. This type of intensification can be done in a post-optimization phase (using the final pool of elite solutions), or periodically during the optimization (using the the current set of elite solutions).

The intensification procedure is executed after each interval of `ifreq` iterations during the optimization. After each intensification phase, if no change in P occurs for at least `ifreq` iterations, the costs of the $|P|/2$ worst solutions in P are set to infinity. This is done to guarantee that the solutions in P are renewed. The $|P|/2$ worst solutions are eventually replaced by solutions generated in the following GRASP with path-relinking iterations. Hence, solutions with a high makespan, but sufficiently different from the solutions in P, are accepted for insertion in the pool.

Table 9.1 Probability estimates of finding a solution at least as good as the target solution, as a function of maximum solution time for GRASP and GP+PR. Instances are abz6, mt10, orb5 and la21, with target values 947, 950, 910 and 1110, respectively.

	abz6		mt10		orb5		la21	
time	GRASP	GP+PR	GRASP	GP+PR	GRASP	GP+PR	GRASP	GP+PR
100s	.09	.39	.03	.01	.42	.67	.10	.10
500s	.48	.93	.19	.71	.92	1.00	.36	.92
1000s	.74	1.00	.37	.97	.98	1.00	.56	1.00
1500s	.84	1.00	.54	1.00	.99	1.00	.69	1.00

Path-relinking as a post-optimization step was introduced in Aiex et al. (2000a). After applying path-relinking between all pairs of elite solutions and no further change in the elite set occurs, the local search procedure of Subsection 9.2.2 is applied to each elite solution, as the solutions produced by path-relinking are not always local optima. The local optima found are candidates for insertion into the elite set. If a change in the elite set occurs, the entire post-processing step is repeated.

9.3.1 GRASP with path-relinking

We describe how we combined path-relinking and GRASP to form a hybrid GRASP with path-relinking. Pseudo-code for the GRASP with path-relinking for JSP is presented in Figure 9.2. Let maxpool be the size of the elite set. The first maxpool GRASP iterations contribute one solution to the elite set per GRASP iteration (line 19). Path-relinking is not done until the pool of elite solutions is full.

GRASP alternates between the two construction procedures described in Section 9.2. Odd numbered iterations use the randomized time-remaining greedy function (line 6), while even iterations use randomized makespan greedy (line 4). The local search used is the one proposed by Taillard (1994) (line 8).

Once the pool of elite solutions is full, the solution S produced by the local search phase of GRASP is tested to verify its quality (line 10), using the quality criteria described in Section 9.3. If S passes the quality test, bidirectional path-relinking is done between S and all elements of a subset $P' \subseteq P$ (lines 11 to 18). After each path-relinking phase, the best solution obtained by path-relinking is tested for inclusion in the elite pool (lines 14 and 16).

Every ifreq GRASP iterations the path-relinking intensification process is carried out (lines 20).

The GRASP with path-relinking loop from line 2 to 23 continues for at most maxitr iterations, but can be terminated when a schedule having a makespan of at most look4 is found (line 22).

Finally, path-relinking post optimization is done on the elite set (line 24).

9.4 PARALLEL GRASP WITH PATH-RELINKING FOR THE JSP

In this section, we describe two parallel implementations of GRASP with path-relinking for the JSP. The first scheme (called non-collaborative) limits communication between

processors only for problem input, detection of process termination, and determination of best overall solution. In addition to the communication allowed in the non-collaborative scheme, the second scheme (called collaborative) allows processes to exchange information regarding their elite sets.

9.4.1 Non-collaborative scheme

We revisit a basic parallelization scheme for GRASP with path-relinking proposed in Aiex et al. (2000a). Figure 9.15 shows pseudo-code for this *multiple independent walks* scheme (Verhoeven and Aarts, 1995).

Our implementation uses message passing for communication between processors. This communication is limited to program initialization and termination. A single process reads the problem data and passes it to the remaining $\texttt{nproc} - 1$ processes. Processes send a message to all others when they either stop upon finding a solution at least as good as the target or complete the maximum number of allotted iterations.

The non-collaborative parallel GRASP with path-relinking is built upon the sequential algorithm of Figure 9.2. Each process executes a copy of the program. We discuss the differences between the sequential algorithm and this parallel algorithm. In line 1 of Figure 9.15, the rank of the process and the number of processes are determined. Each GRASP construction phase is initialized with a random number generator seed. To assure independence of processes, identical seeds of the random number generator (`rand()`) must not be used by more than one process. The initial seed for process `my_rank` is computed in lines 2 to 4. This way, each process has a sequence of `maxitr` initial seeds.

The **for** loop from line 6 to line 37 executes the iterations. The construction, local search, and path-relinking phases are identical to the those of the sequential algorithm. In line 26, if a process finds a schedule with makespan not greater than `look4`, it sends a flag to each of the other processes indicating that it has found the solution. Likewise, when a process completes `maxitr` iterations, it sends a flag to each of the other processes indicating that it has completed the preset number of iterations (lines 27 to 30).

In line 31, the process checks if there are any status flags to be received. If there is a flag indicating that a schedule with makespan not greater than `look4` has been found, then execution of **for** loop from line 6 to line 37 is terminated (line 33). If a flag indicating that some process has completed the preset number of iterations, a counter (*num_stop*) of the number of processes that have completed the iterations is incremented (line 34). If all processes have completed their iterations, the execution of the **for** loop is terminated (line 36).

Each process, upon terminating **for** loop going from line 6 to line 37, runs the post-optimization phase on the pool of elite solutions (line 38). A reduce operator (`GET_GLOBAL_BEST`) determines the global best solution among all processes in line 39 and returns this solution.

A parallel pure GRASP can be obtained from the algorithm in Figure 9.15 by skipping the execution of lines 13 to 23. As in a basic GRASP, it is necessary to keep track of the best solution found and no pool handling operations are necessary. Therefore,

Table 9.2 Time to find a solution with cost at least as good as the target value, as a function of probability. Problem mt10 was tested for target values 970, 960 and 950. The percentage reduction in solution time of GP+PR with respect of GRASP is shown for each target value.

	look4=970			look4=960		
prob.	GRASP	GP+PR	red.(%)	GRASP	GP+PR	red.(%)
0.2	44.92s	144.61s	-221.92	214.94s	198.75s	7.53
0.5	163.93s	211.66s	-29.11	667.41s	295.71s	55.69
0.8	386.94s	292.98s	24.28	1362.65s	416.35s	69.44

	look4=950		
prob.	GRASP	GP+PR	red.(%)
0.2	546.45s	263.02s	51.86
0.5	1422.20s	394.51s	72.26
0.8	2951.01s	578.53s	80.39

intensification and post-optimization are not carried out during a pure GRASP parallel approach.

9.4.2 Collaborative scheme

In the collaborative parallel GRASP with path-relinking, processes share elite set information. We now describe this scheme, whose pseudo-code is presented in Figure 9.16. This algorithm is built on top of the non-collaborative scheme presented in the previous subsection. We limit our discussion to the differences between the two schemes.

The differences between the non-collaborative and collaborative schemes occur in the path-relinking phase. Before doing path-relinking between solutions S and T, each process checks if one or more other processes have sent new elite solutions to it. If there are new elite solutions to be received, RECEIVE_SOLUTIONS (in lines 17 and 21) receives the elite solutions, tests if each elite solution can be accepted for insertion into its local elite set, and inserts any accepted elite solution. Upon termination of each path-relinking leg, if the local elite set is updated, then (in lines 20 and 24) the process writes the new elite set solutions to a local send buffer. In line 26, if the local send buffer is not empty, the process sends the buffer contents to the other processes.

Another difference between the non-collaborative and the collaborative schemes concerns the INTENSIFY procedure. In the collaborative scheme, whenever the local elite set pool is updated, the new elite set solutions are written to the send buffer. These bufferized solutions will be sent to the other processes the next time that procedure SEND_SOLUTIONS is invoked.

9.5 COMPUTATIONAL RESULTS

This section reports on results of computational experiments done with sequential and parallel versions of the pure GRASP and GRASP with path-relinking heuristics proposed in this paper.

Table 9.3 Probability estimates of finding a solution at least as good as the target solution, as a function of maximum solution time for GP+C−INT and GP+PR. Instances are abz6, mt10, orb5 and la21, with target values 965, 960, 930 and 1130, respectively.

	abz6		mt10		orb5	
time	GP+C−INT	GP+PR	GP+C−INT	GP+PR	GP+C−INT	GP+PR
100s	.49	1.00	.04	.02	.16	.95
500s	.94	1.00	.17	.90	.46	1.00
1000s	1.00	1.00	.27	1.00	.75	1.00
1500s	1.00	1.00	.34	1.00	.86	1.00

	la21	
time	GP+PR	GP+C−INT
100s	.09	.53
500s	.37	1.00
1000s	.55	1.00
1500s	.69	1.00

Table 9.4 Experimental results on problem classes abz, car, mt, and orb. Table shows problem name, problem dimension (jobs and machines), the best known solution (BKS), the best solution found by GP+C−INT, total number of GP+PR iterations performed, CPU time per 1000 GP+PR iterations, the best solution found by GP+PR, and the relative percentage error of the GP+PR solution with respect to the BKS.

problem	$\lvert\mathcal{J}\rvert$	$\lvert\mathcal{M}\rvert$	BKS	GP+C−INT	GP+PR: itrs ($\times 10^6$)	time (10^3 iter)	sol.	error (%)
abz5	10	10	1234	1238	1.0	2.53s	1234	0.0
abz6	10	10	943	947	0.3	2.26s	943	0.0
abz7	15	20	665	723	50.0	11.66s	692	4.1
abz8	15	20	670	729	10.0	49.78s	705	5.2
abz9	15	20	691	758	1.0	875.92s	740	7.1
car1	11	5	7038	7038	0.001	15.31s	7038	0.0
car2	13	4	7166	7166	0.001	52.14s	7166	0.0
car3	12	5	7312	7366	50.7	2.08s	7312	0.0
car4	14	4	8003	8003	0.01	2.79s	8003	0.0
car5	10	6	7702	7702	0.5	4.40s	7702	0.0
car6	8	9	8313	8313	0.01	11.85s	8313	0.0
car7	7	7	6558	6558	0.001	16.05s	6558	0.0
car8	7	7	8264	8264	0.02	4.66s	8264	0.0
mt06	6	6	55	55	0.00001	1.74s	55	0.0
mt10	10	10	930	938	2.5	4.05s	930	0.0
mt20	20	5	1165	1169	4.5	46.48s	1165	0.0
orb1	10	10	1059	1070	1.2	46.75s	1059	0.0
orb2	10	10	888	889	1.1	11.45s	888	0.0
orb3	10	10	1005	1021	6.5	33.32s	1005	0.0
orb4	10	10	1005	1031	100.0	1.94s	1011	0.6
orb5	10	10	887	891	20.0	14.61s	889	0.2
orb6	10	10	1010	1013	3.5	43.75s	1012	0.2
orb7	10	10	397	397	0.03	18.72s	397	0.0
orb8	10	10	899	909	1.6	24.26s	899	0.0
orb9	10	10	934	945	11.1	4.38s	934	0.0
orb10	10	10	944	953	0.3	33.25s	944	0.0

Table 9.5 Experimental results on problem class la (problems la01 to la20). Table shows problem name, problem dimension (jobs and machines), the best known solution (BKS), the best solution found by GP+C−INT, total number of GP+PR iterations performed, CPU time per 1000 GP+PR iterations, the best solution found by GP+PR, and the relative percentage error of the GP+PR solution with respect to the BKS.

problem	$\|\mathcal{J}\|$	$\|\mathcal{M}\|$	BKS	GP+C−INT	GP+PR: itrs ($\times 10^6$)	time (10^3 iter)	sol.	error (%)
la01	10	5	666	666	0.0001	0.82s	666	0.0
la02	10	5	655	655	0.004	2.74s	655	0.0
la03	10	5	597	604	0.01	2.95s	597	0.0
la04	10	5	590	590	0.001	15.71s	590	0.0
la05	10	5	593	593	0.0001	1.09s	593	0.0
la06	15	5	926	926	0.0001	8.00s	926	0.0
la07	15	5	890	890	0.0001	1.58s	890	0.0
la08	15	5	863	863	0.0003	5.49s	863	0.0
la09	15	5	951	951	0.0001	3.40s	951	0.0
la10	15	5	958	958	0.0001	11.50s	958	0.0
la11	20	5	1222	1222	0.0001	3.14s	1222	0.0
la12	20	5	1039	1039	0.0001	2.89s	1039	0.0
la13	20	5	1150	1150	0.0001	3.20s	1150	0.0
la14	20	5	1292	1292	0.0001	5.70s	1292	0.0
la15	20	5	1207	1207	0.0002	122.75s	1207	0.0
la16	10	10	945	946	1.3	2.27s	945	0.0
la17	10	10	784	784	0.02	3.29s	784	0.0
la18	10	10	848	848	0.05	9.07s	848	0.0
la19	10	10	842	842	0.02	15.14s	842	0.0
la20	10	10	902	907	17.0	1.63s	902	0.0

9.5.1 Computer environment

The experiments were done on an SGI Challenge computer (16 196-MHz MIPS R10000 processors and 12 194-MHz R10000 processors) with 7.6 Gb of memory. Each run of the sequential implementations used a single processor. The parallel implementations were run on 1, 2, 4, 8, and 16 processors. Load on the machine was low throughout the experiments and therefore processors were always available.

The algorithms were coded in Fortran and were compiled with the SGI MIPSpro F77 compiler using flags `-O3 -r4 -64`. The Message-Passing Interface (MPI) specification has become a common standard for message-passing libraries for parallel computations (Snir et al., 1998). The parallel codes used SGI's Message Passing Toolkit 1.4, which contains a fully compliant implementation of the MPI 1.2 specification. CPU times for the sequential implementation were measured with the system function `etime`. In the parallel experiments, times measured were wall clock times, and were done with the MPI function `MPI_WT`. This is also the case for runs with a single processor that are compared to 2, 4, 8, and 16 parallel processor runs. Timing in the parallel runs excludes the time to read the problem data, initialize the random number generator seeds, and to output the solution.

Table 9.6 Experimental results on problem class la (problems la21 to la40). Table shows problem name, problem dimension (jobs and machines), the best known solution (BKS), the best solution found by GP+C−INT, total number of GP+PR iterations performed, CPU time per 1000 GP+PR iterations, the best solution found by GP+PR, and the relative percentage error of the GP+PR solution with respect to the BKS.

problem	$\|\mathcal{J}\|$	$\|\mathcal{M}\|$	BKS	GP+C−INT	GP+PR: itrs ($\times 10^6$)	time (10^3 iter)	sol.	error (%)
la21	15	10	1047	1091	100.0	3.51s	1057	1.0
la22	15	10	927	960	26.0	3.45s	927	0.0
la23	15	10	1032	1032	0.01	39.39s	1032	0.0
la24	15	10	935	978	125.0	3.26s	954	2.0
la25	15	10	977	1028	32.0	3.29s	984	0.7
la26	20	10	1218	1271	3.5	6.35s	1218	0.0
la27	20	10	1235	1320	10.5	27.51s	1269	2.8
la28	20	10	1216	1293	20.0	17.77s	1225	0.7
la29	20	10	1157	1293	50.0	6.17s	1203	4.0
la30	20	10	1355	1368	3.0	7.61s	1355	0.0
la31	30	10	1784	1784	0.01	267.60s	1784	0.0
la32	30	10	1850	1850	0.0001	12.66s	1850	0.0
la33	30	10	1719	1719	0.001	875.11s	1719	0.0
la34	30	10	1721	1753	0.05	80.33s	1721	0.0
la35	30	10	1888	1888	0.01	348.32s	1888	0.0
la36	15	15	1268	1334	51.0	5.54s	1287	1.5
la37	15	15	1397	1457	20.0	12.51s	1410	0.9
la38	15	15	1196	1267	20.0	32.87s	1218	1.8
la39	15	15	1233	1290	6.0	59.06s	1248	1.2
la40	15	15	1222	1259	2.0	104.18s	1244	1.8

Table 9.7 Experimental results: Overall solution quality by problem class. Sum of all best known solutions (BKS) for each class is compared with sum of best GP+PR solutions. Relative error is of GP+PR solution with respect to BKS.

problem	sum of BKS	sum of GP+PR sol.	relative error (%)
abz	4203	4314	2.64
car	60356	60356	0.00
mt	2150	2150	0.00
orb	9028	9038	0.11
la	44297	44513	0.49

9.5.2 Test Problems

The experiments were done on 66 instances from five classes of standard JSP test problems: abz, car, la, mt, and orb. The problem dimensions vary from 6 to 30

Table 9.8 Experimental results: Percentage of GP+PR solutions within a tolerance of the best known solution (BKS).

	tolerance					
problem	0%	.5%	1%	2%	5%	10%
abz	40.0	40.0	40.0	40.0	60.0	100.0
car	100.0	100.0	100.0	100.0	100.0	100.0
mt	100.0	100.0	100.0	100.0	100.0	100.0
orb	70.0	90.0	100.0	100.0	100.0	100.0
la	72.5	72.5	82.5	95.0	100.0	100.0

jobs and from 4 to 20 machines. All instances tested were downloaded from Beasley's OR-Library [1] (Beasley, 1990).

9.5.3 The sequential experiments

The goal of the sequential experiments was to observe the general behavior of the implementations of the proposed algorithms. In these experiments, we present results comparing the following heuristics:

1. GRASP: Pure GRASP alternating between makespan and time-remaining randomized greedy constructions;
2. GP+PR: GRASP with path-relinking described in Section 9.3;
3. GP+C-INT: The GRASP with construction intensification and POP, described in Binato et al. (2001).

We aim to verify how the solutions obtained by GP+PR compare to the best known solutions for a set of standard test problems. To illustrate the effectiveness of the proposed hybrid GRASP, the solution times to target solution of GP+PR are compared to the solution times of GRASP and GP+C-INT.

On all sequential (and parallel) implementations tested in this paper, the restricted candidate list parameter α is chosen at random from the uniform distribution in the interval $[0, 1]$ at each GRASP iteration and remains fixed throughout the iteration.

For the experiments performed with GP+PR, we used a pool of size $|P|$=30 and a differentiation factor for insertion into the pool of *dif*=25%. In all experiments done with GP+PR, path-relinking was applied between the solution obtained by GRASP and all solutions in the pool. The standard deviation used to verify if a solution obtained by the local search will take part in path-relinking is computed from the costs of the first n=10,000 GRASP solutions. In the runs of GP+PR, used to generate the plots shown in this paper, intensification and post-optimization are not applied. In the runs of GP+PR shown in Tables 9.4, 9.5, and 9.6, the intensification is applied after each interval of *freq*=500,000 iterations.

In all experiments performed with GP+C-INT, the program was configured to use the same parameter values used in the tests reported in Binato et al. (2001). Therefore,

[1] http://mscmga.ms.ic.ac.uk/jeb/orlib/jobshopinfo.html

a pool of 30 elite solutions was used in the intensification phase and a differentiation factor of *dif*=80% was used to control the insertion into the pool. POP was activated with a parameter `freq` = 40, i.e., it was applied after the construction of 40% and 80% of the partial solution. A linear distribution function was used to bias the selection of candidate elements in the RCL.

To study the effect of path-relinking on GRASP, we compared `GRASP` and `GP+PR` on problems `abz6`, `mt10`, `orb5`, and `la21`. Two hundred independent runs of the two heuristics were done for each of the four problems. Execution was interrupted when a solution with cost at least as good as `look4` was found. The `look4` values used for problems `abz6`, `mt10`, `orb5`, and `la21` were 947, 950, 910, and 1110, respectively. These values are far from the optimal values, and in general, can be obtained after a few iterations. Figure 9.3 shows the empirical distributions for solution time of `GRASP` and `GP+PR`. To plot these empirical distributions, we associate with the i-th sorted running time (t_i) a probability $p_i = (i - \frac{1}{2})/200$. The points $z_i = (t_i, p_i)$ are then plotted for $i = 1, \ldots, 200$. We observe in the plots that `GP+PR` finds the target solution faster than `GRASP`. Table 9.1 shows estimated probabilities of finding the target solution as a function of CPU time for the four instances tested. For example, for a computational time of at most 500 seconds, the estimated probability of finding a solution at least as good as the target solution for problem `abz6` is 93% for `GP+PR`, while for `GRASP` it is 48%. On problem `la21`, the estimated probability of finding a solution at least as good as the target solution in time at most 1000 seconds is 56% for `GRASP` and 100% for `GP+PR`. These results illustrate for the JSP, the fact observed in Aiex et al. (2000a), that although each iteration of a hybrid approach of GRASP with path-relinking takes more computational time when compared to an iteration of a pure GRASP, it is compensated by the reduced number of iterations needed to find the target solution.

Figure 9.4 shows a comparison between `GRASP` and `GP+PR` for problem `mt10`, using three target values: 970, 960, and 950. These target values are 4.3%, 3.2%, and 2.1% away from the best known value, respectively. The figure is composed of three plots, where the difficulty of obtaining the target solution grows from top to bottom. The empirical distributions are plotted the same way as in the plots of Figure 9.3. For the topmost plot, we observe that `GRASP` finds the target solution before `GP+PR` for probabilities below 68%. This occurs because the target value sought is easy to find and `GRASP` iterations are faster than `GP+PR` iterations. By gradually increasing the difficulty to find the target values on the two remaining plots (middle and bottom), we observe that the probabilities for which `GRASP` still finds the target solution faster than `GP+PR` decrease to 19% and 9%, respectively. Table 9.2 shows the computational times for `GRASP` and `GP+PR` to find each of the three target values with probabilities 20%, 50%, and 80%. For each pair of GRASP variant and target value, the table also shows the percentage reduction in solution time of `GP+PR` with respect to the solution time of `GRASP`, as a function of the probability. The difficulty to obtain the target solution grows from left to right and top to bottom in the table. We observe, that as the target value approaches the best known value, the percentage reduction in solution time of `GP+PR` grows. For example, for a target value of 960, and a probability of 50%, the percentage reduction in solution time is 55.6%. Decreasing the target value to 950,

the percentage reduction in solution time increases to 72.3% for the same probability of 50%.

Variants GP+PR and GP+C-INT are compared in Figure 9.5. The empirical distributions are plotted for these GRASPs using the same methodology used to plot the empirical distributions in Figures 9.3 and 9.4. The same test problems, abz6, mt10, orb5, and la21, are used in this experiment, with target values 965, 960, 930, and 1130, respectively. Notice that the target values used in this experiment are easier to obtain than the target values used to compare GRASP and GP+PR for the same four instances. This was necessary because of the high computational time needed for GP+C-INT to obtain target solutions of quality comparable to the quality of the solutions found in the first experiment. Table 9.3 shows, for GP+C-INT and GP+PR, estimates of probabilities of finding a solution with cost at least as good as the target value, as a function of maximum solution time. For example, for problem la21, we observe that the estimated probability for GP+C-INT to obtain a solution with cost at most 1130 in less than 500 seconds is 37%, while for GP+PR this probability is 100%. For problem mt10, we observe that the estimated probabilities for GP+C-INT and GP+PR to find the target solution in less than 1000 seconds are 27% and 100%, respectively. Therefore, we verify that the use of intensification in a GRASP shows better results when this phase is carried out after the local search phase, i.e., after a pure GRASP iteration. This happens because a premature intensification, i.e., an intensification phase done during the GRASP construction phase, might reduce drastically the number of local minima visited during a run of GRASP.

To verify the behavior of the proposed algorithm in terms of solution quality, GP+PR was extensively executed for all test problems considered. The number of GRASP iterations was frequently in the millions (where each GRASP iteration uses a different seed of the random number generator). These results are shown in Tables 9.4, 9.5, and 9.6. Each table shows problem name, problem dimension (number of jobs and machines), the best known solution (BKS), the cost of the solution found by GP+C-INT, and, for GP+PR, the total number of GRASP iterations executed, CPU time in seconds to run 1000 GRASP iterations, the cost of the best solution found, and the percentage relative error of the GP+PR solution with respect to the BKS.

Of the 66 tested instances, GP+PR found the BKS in 49 cases (74.2%). It found a solution within 0.5% of the BKS for 50 instances (75.7%). In 56 instances (84.8%), GP+PR solution was within 1% of the BKS and in 61 cases (92.4%) it was within 2% of the BKS. GP+PR solution was within 5% of the BKS in 64 instances (97%), while for all other cases, the solution found was within 7.5% of the BKS.

Tables 9.7 and 9.8 summarizes the results for each problem class. Table 9.7 shows, for each problem class, its name, the sum of the BKS values, the sum of the values of the best solutions found by GP+PR, and the percentage relative error of the sum of the values of the best GP+PR solutions with respect to the sum of the BKS values. Table 9.8 shows, for each problem class, its name, and the percentage of instances for which a GP+PR solution within 0%, 0.5%, 1%, 2%, 5%, and 10% of the BKS was produced. From these tables, one can conclude that the easiest classes are car and mt, for which GP+PR obtained the BKS for all instances. For classes orb and la, the average relative errors are within 0.5% of the BKS and therefore, GP+PR was capable of producing

solutions of high quality for most problems in these classes. The most difficult class was abz, where the average relative error with respect to the BKS achieved 2.64%.

9.5.4 Probability distribution for solution time

Aiex et al. (2000b) studied the empirical probability distributions of the random variable *time to target solution* in five GRASP implementations. They showed that, given a target solution value, the time it takes GRASP to find a solution at least as good as the target fits a two-parameter exponential distribution. Standard methodology for graphical analysis (Chambers et al., 1983) was used to compute the empirical and theoretical distributions and to estimate the parameters of the distributions. We use the same methodology to study *time to target value* for GRASP and GP+PR. Our objective is to show that these variants of GRASP have time to target value distributions that fit a two-parameter exponential distribution.

The quantile-quantile plots (Q-Q plots) and the plots showing the empirical and theoretical distributions of the random variable *time to target solution* for GRASP are shown in Figures 9.6 and 9.7, respectively. Analogously, Figures 9.8 and 9.9 show the Q-Q plots and the plots with the empirical and theoretical distributions of the random variable *time to target solution* for GP+PR. Three target values are considered for each of the test problems, abz6, mt10, orb5, and la21, for the two GRASP variants. All plots are computed with 200 runs of the GRASP variant. For each of the 200 runs of each combination, the random number generator is initialized with a distinct seed and therefore the runs are independent.

Figures 9.6 and 9.8 are made up of 12 quantile-quantile plots, one for each pair of problem instance/target value for GRASP and GP+PR, respectively. Analogously, Figures 9.7 and 9.9 are made up of 12 plots showing the empirical and theoretical distributions of the random variable *time to target solution*, each corresponding to a pair of problem instance/target value for GRASP and GP+PR, respectively. Each figure is made up of four rows, each corresponding to a different problem. Each row of the figure depicts three plots, where the difficulty to find the target value increases from left to right. Our description of each plot follows Aiex et al. (2000b) closely. For each instance/variant pair, the running times are sorted in increasing order. To plot the empirical distribution, we associate with the i-th sorted running time (t_i) a probability $p_i = (i - \frac{1}{2})/200$, and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 200$.

Tables 9.10 and 9.13 show the target values and the parameters estimated by the methodology for GRASP and GP+PR, respectively. Following the methodology proposed in Chambers et al. (1983), we first draw the theoretical quantile-quantile plot for the data to estimate the parameters of the two-parameter exponential distribution. To describe Q-Q plots, recall that the cumulative distribution function for the two-parameter exponential distribution is given by

$$F(t) = 1 - e^{-(t-\mu)/\lambda},$$

where λ is the mean of the distribution data (and indicates the spread of the data) and μ is the shift of the distribution with respect to the ordinate axis.

For each value p_i, $i = 1, \ldots, 200$, we associate a p_i-quantile $Qt(p_i)$ of the theoretical distribution. For each p_i-quantile we have, by definition, that

$$F((Qt(p_i)) = p_i.$$

Hence, $Qt(p_i) = F^{-1}(p_i)$ and therefore, for the two-parameter exponential distribution, we have

$$Qt(p_i) = -\lambda \ln(1 - p_i) + \mu.$$

The quantiles of the data of an empirical distribution are simply the (sorted) raw data.

A theoretical quantile-quantile plot (or theoretical Q-Q plot) is obtained by plotting the quantiles of the data of an empirical distribution against the quantiles of a theoretical distribution. This involves three steps. First, the data (in our case, the measured times) are sorted in ascending order. Second, the quantiles of the theoretical exponential distribution are obtained. Finally, a plot of the data against the theoretical quantiles is made.

In a situation where the theoretical distribution is a close approximation of the empirical distribution, the points in the Q-Q plot will have a nearly straight configuration. If the parameters λ and μ of the theoretical distribution that best fits the measured data could be estimated a priori, the points in a Q-Q plot would tend to follow the line $x = y$. Alternatively, in a plot of the data against a two-parameter exponential distribution with $\lambda' = 1$ and $\mu' = 0$, the points would tend to follow the line $y = \lambda x + \mu$. Consequently, parameters λ and μ of the two-parameter exponential distribution can be estimated, respectively, by the slope and intercept of the line depicted in the Q-Q plot.

To avoid possible distortions caused by outliers, we do not estimate the distribution mean by linear regression on the points of the Q-Q plot. Instead, we estimate the slope $\hat{\lambda}$ of line $y = \lambda x + \mu$ using the upper quartile q_u and lower quartile q_l of the data. The upper and lower quartiles are, respectively, the $Q(\frac{1}{4})$ and $Q(\frac{3}{4})$ quantiles, respectively. We take

$$\hat{\lambda} = (z_u - z_l)/(q_u - q_l)$$

as an estimate of the slope, where z_u and z_l are the u-th and l-th points of the ordered measured times, respectively. This informal estimation of the distribution of the measured data mean is robust since it will not be distorted by a few outliers (Chambers et al., 1983). These estimates are used to plot the theoretical distributions on the plots on the left side of the figures.

To analyze the straightness of the Q-Q plots, we superimpose them with variability information. For each plotted point, we show plus and minus one standard deviation in the vertical direction from the line fitted to the plot. An estimate of the standard deviation for point z_i, $i = 1, \ldots, 200$, of the Q-Q plot is

$$\hat{\sigma} = \hat{\lambda} \sqrt{\frac{p_i}{(1 - p_i)200}}.$$

Figures 9.6 and 9.8 show that there is little departure from straightness in the Q-Q plots for GRASP, as well as for GP+PR. We also observe that as the difficulty of finding

Table 9.9 Speedup with respect to a single processor implementation and efficiency (speedup divided by number of processors). Algorithm is the parallel implementation of GRASP. Instances are abz6, mt10, orb5, and la21, with target values 960, 960, 920, and 1120, respectively.

	number of processors							
	2		4		8		16	
problem	speedup	eff.	speedup	eff.	speedup	eff.	speedup	eff.
abz6	2.04	1.02	4.75	1.18	8.87	1.10	19.17	1.19
mt10	1.62	.81	4.07	1.01	7.34	.91	14.81	.92
orb5	2.12	1.06	3.97	.99	7.63	.95	14.10	.88
la21	1.94	.97	4.98	1.24	8.13	1.01	19.63	1.22
average:	**1.93**	**.96**	**4.44**	**1.10**	**7.99**	**.99**	**16.92**	**1.05**

the target value increases, the plotted points become more fitted to the estimated line. Therefore, we verify that the distributions fit a two-parameter exponential distribution.

Binato et al. (2001) show that the probability distribution of solution time of GP+C−INT fits a two-parameter exponential distribution. In this section, we show that the probability distributions of solution time of a GRASP where the construction phase is computed alternating between two greedy functions (GRASP) and of a GRASP with path-relinking restricted to iterations where the local search obtained high quality solutions (GP+PR) also fit a two-parameter exponential distribution. These results reinforce the conclusions drawn in Aiex et al. (2000b) for pure GRASPs.

The following can be stated for a two parameter (shifted) exponential distribution (Aiex et al., 2000b; Verhoeven and Aarts, 1995). Let $P_\rho(t)$ be the probability of not having found a given (target) solution in t time units with ρ independent processes. If $P_1(t) = e^{-(t-\mu)/\lambda}$ with $\lambda \in \mathbb{R}^+$ and $\mu \in \mathbb{R}$, i.e. P_1 corresponds to a two parameter exponential distribution, then $P_\rho(t) = e^{-\rho(t-\mu)/\lambda}$. This follows from the definition of the two-parameter exponential distribution. It implies that the probability of finding a solution of a given value in time ρt with a sequential process is equal to $1 - e^{-(\rho t-\mu)/\lambda}$ while the probability of finding a solution at least as good as that given value in time t with ρ independent parallel processes is $1 - e^{-\rho(t-\mu)/\lambda}$. Note that if $\mu = 0$, then both probabilities are equal and correspond to the non-shifted exponential distribution. Furthermore, if $\rho\mu \ll \lambda$, then the two probabilities are approximately equal and it is possible to approximately achieve linear speed-up in solution time to target solution by multiple independent processes.

9.5.5 The parallel experiments

The parallel algorithms used in these experiments are:

1. the pure GRASP;
2. the non-collaborative GRASP with path-relinking;
3. the collaborative GRASP with path-relinking.

In these experiments, we disable stopping due to maximum number of iterations, i.e. the algorithms terminate only when a solution of value at least as good as look4 is

Table 9.10 Test problems used to study the empirical probability distributions of the random variable time to target solution of GRASP. Table shows for each tested problem, cost of the BKS, target value and estimated parameters.

			estimated parameters	
problem	BKS	target	$\hat{\mu}$	$\hat{\lambda}$
abz6	943	970	0.203	3.804
		960	0.428	15.567
		950	-1.323	68.490
mt10	930	970	-9.403	233.092
		960	11.723	885.034
		950	109.528	1827.882
orb5	887	930	-0.783	15.757
		920	1.249	38.273
		910	-1.011	191.111
la21	1047	1130	-4.115	50.343
		1120	-1.015	206.836
		1110	-87.594	1268.081

Table 9.11 Estimates of probability of finding a solution at least as good as the target solution in a given running time, as a function of number of processors. Algorithm is the parallel implementation of GRASP. Instances are abz6, mt10, orb5, and la21, with target values 960, 960, 920, and 1120, respectively.

		probab. parallel GRASP number of processors				
problem	time	1	2	4	8	16
abz6	10s	.34	.67	.93	1.00	1.00
	20s	.61	.90	.98	1.00	1.00
	50s	.90	.98	1.00	1.00	1.00
mt10	10s	.04	.02	.04	.07	.19
	100s	.16	.20	.45	.64	.82
	500s	.46	.60	.92	1.00	1.00
orb5	10s	.20	.37	.62	.87	.99
	20s	.36	.62	.84	.96	1.00
	50s	.70	.94	1.00	1.00	1.00
la21	10s	.04	.08	.18	.30	.54
	100s	.29	.57	.87	.96	1.00
	500s	.89	.97	1.00	1.00	1.00

Table 9.12 Speedup with respect to a single processor implementation. Algorithms are independent and cooperative implementations of GP+PR. Instances are abz6, mt10, orb5, and la21, with target values 943, 938, 895, and 1100, respectively.

	speedup independent (number of processors)				speedup cooperative (number of processors)			
problem	2	4	8	16	2	4	8	16
abz6	2.00	3.36	6.44	10.51	2.40	4.21	11.43	23.58
mt10	1.57	2.12	3.03	4.05	1.75	4.58	8.36	16.97
orb5	1.95	2.97	3.99	5.36	2.10	4.91	8.89	15.76
la21	1.64	2.25	3.14	3.72	2.23	4.47	7.54	11.41
average:	**1.79**	**2.67**	**4.15**	**5.91**	**2.12**	**4.54**	**9.05**	**16.93**

Table 9.13 Test problems used to study the empirical probability distributions of the random variable time to target solution of GP+PR. Table shows for each tested problem, cost of the BKS, target value and estimated parameters.

			estimated parameters	
problem	BKS	target	$\hat{\mu}$	$\hat{\lambda}$
abz6	943	950	25.067	40.348
		947	30.652	140.487
		943	92.220	744.247
mt10	930	950	206.950	249.865
		940	255.666	334.774
		938	305.281	524.236
orb5	887	910	47.322	44.268
		900	82.006	200.680
		895	131.435	418.053
la21	1047	1110	140.530	155.441
		1105	140.399	248.812
		1100	181.539	390.571

Table 9.14 Estimates of probability of finding a solution at least as good as the target solution in a given running time, as a function of number of processors. Algorithms are independent and cooperative implementations of GP+PR. Instances are abz6, mt10, orb5, and la21, with target values 943, 938, 895, and 1100, respectively.

		probab. independent (number of processors)					probab. cooperative (number of processors)				
problem	time	1	2	4	8	16	1	2	4	8	16
abz6	100s	.01	.03	.12	.25	.47	.01	.12	.24	.64	.94
	500s	.30	.59	.79	.95	1.00	.30	.61	.79	1.00	1.00
	1000s	.57	.85	.97	.99	1.00	.57	.92	.98	1.00	1.00
mt10	100s	0.0	0.0	0.0	.02	0.05	0.0	0.0	.05	.34	.82
	500s	.17	.32	.55	.82	.98	.17	.49	.95	1.00	1.00
	1000s	.54	.79	.95	1.00	1.00	.54	.85	1.00	1.00	1.00
orb5	100s	0.0	0.0	.02	.03	.19	0.0	.07	.42	.74	.92
	500s	.35	.75	.93	1.00	1.00	.35	.80	.97	1.00	1.00
	1000s	.75	.97	1.00	1.00	1.00	.75	.95	1.00	1.00	1.00
la21	100s	0.0	0.0	0.0	.02	.06	0.0	.02	.08	.44	.87
	500s	.29	.52	.82	.98	1.00	.29	.79	1.00	1.00	1.00
	1000s	.75	.98	1.00	1.00	1.00	.75	.98	1.00	1.00	1.00

found. The parallel GRASP was studied for problems abz6, mt10, orb5, and la21, with look4 values 960, 960, 920, and 1120, respectively. The independent and cooperative parallel implementations of GP+PR were also tested for problems abz6, mt10, orb5, and la21, but with more difficult look4 values 943, 938, 895, and 1100, respectively. The parameters of the procedures used in the parallel approaches were the same used for testing the sequential algorithm. Intensification and post-optimization are not carried out during the experiments with the parallel implementations. Figure 9.10 shows speedup and empirical distributions for the parallel implementations of GRASP. Analogously, Figures 9.11, 9.12, 9.13, and 9.14 show speedup and empirical distributions for both parallel implementations of GP+PR. The plots were generated

with 60 independent runs for each number of processors considered (1, 2, 4, 8, and 16 processors).

Table 9.9 summarizes the speedups shown in the plots. The table also shows efficiency (speedup divided by number of processors) values. Speedups are on average approximately linear. Table 9.10 shows the values of the parameters μ and λ of the two-parameter exponential distributions plotted for the pairs of instance/target values used to study the behavior of GRASP. The parameter μ is an estimate of the minimum time needed for GRASP to find the target value for the instances. The parameter λ is an estimate of the spread of the measured times for pair of instance/target value. The sum $\mu+\lambda$ is an estimate of the average solution time for pair of instance/target value. For a small value of parameter μ, a two-parameter exponential distributions can be approximated by a simple exponential distribution. Therefore, approximate linear speedups were expected for the parallel GRASP on this set of instance/target values.

Table 9.11 shows, for given running times, the estimated probability of finding a solution at least as good as the target solution in that time, as a function of number of processors. The table shows, for example, that the probability of finding a solution of value at most 920 on problem orb5 in at most 10 seconds, goes from 20% with one processor, to 62% with four processors, and to 99% with sixteen processors.

Table 9.12 summarizes the speedups shown in the plots for the independent and cooperative parallel approaches of GP+PR. Sublinear speedups are observed for the independent approach. Table 9.13 shows the values of parameters μ and λ of the two-parameter exponential distributions plotted for the pairs of instance/target values used to study GP+PR. We notice that the ratios λ/μ computed with the parameters in this table are much lower than the values of λ/μ, for the parameters estimated for the pairs of instance/target values used to study GRASP. As stated before, although GP+PR finds the target solution faster than GRASP, its iterations need higher CPU times, which corresponds to higher values of μ. Path-relinking also speedups GRASP, reducing the spread of the solution time, i.e., the parameter λ. Therefore, μ values are higher and λ values are lower for GP+PR with respect to GRASP parameters. For these reasons, the distributions plotted for GP+PR cannot be approximated by a simple exponential distribution. As noted in the observation about the two-parameter exponential distribution, as the number of used processors ρ increases, the speedup of the algorithm degrades. That observation does not take into account sharing of information by the processes. Therefore, no conclusions from the distributions plotted for the sequential GP+PR can be drawn for the cooperative approach. However, we observe an approximate linear speedup for all instances tested for the cooperative approach, outperforming the independent variant.

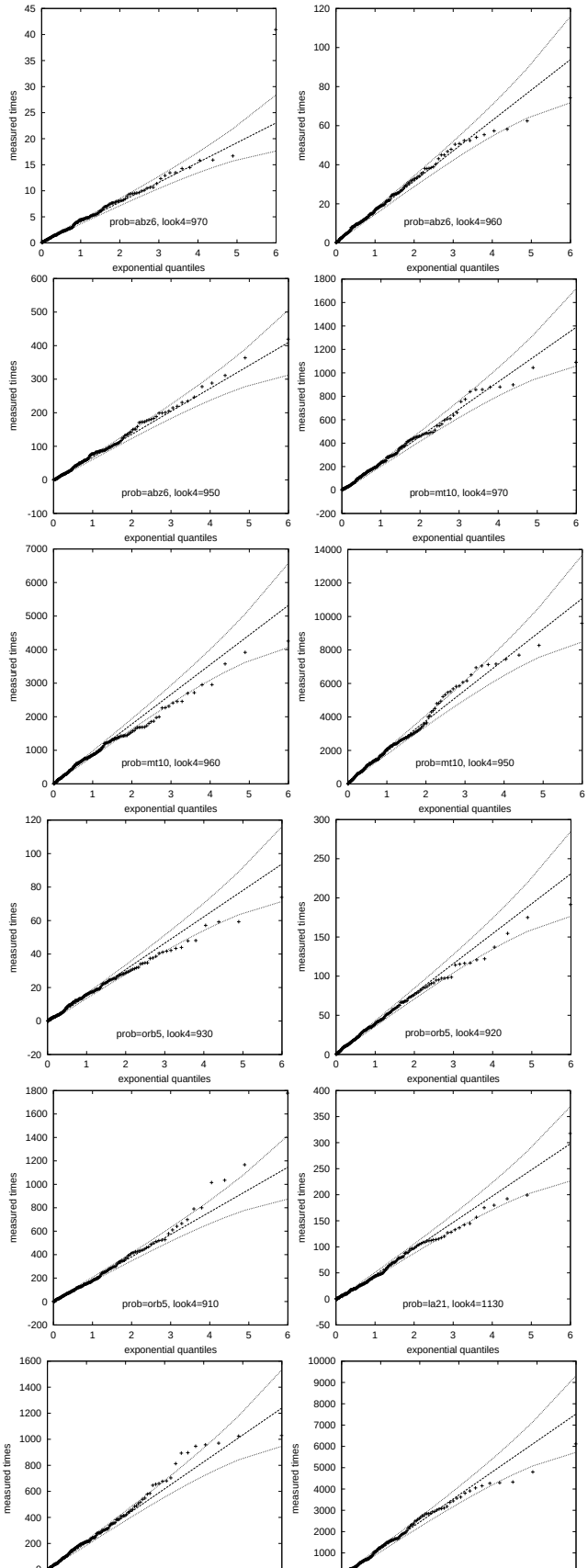
In Table 9.14, the estimated probability of finding a solution at least as good as the target solution before a specified time is shown as a function of number of processors. For example, the table shows, for problem mt10, that the probability of finding a solution of value at least as good as 938 in at most 500 seconds, goes from 32% with two processor, to 55% with four processors, and to 98% with 16 processors, on the independent approach. For the cooperative approach, these values increase to 49%, 95% and 100%, for two, four and sixteen processors, respectively.

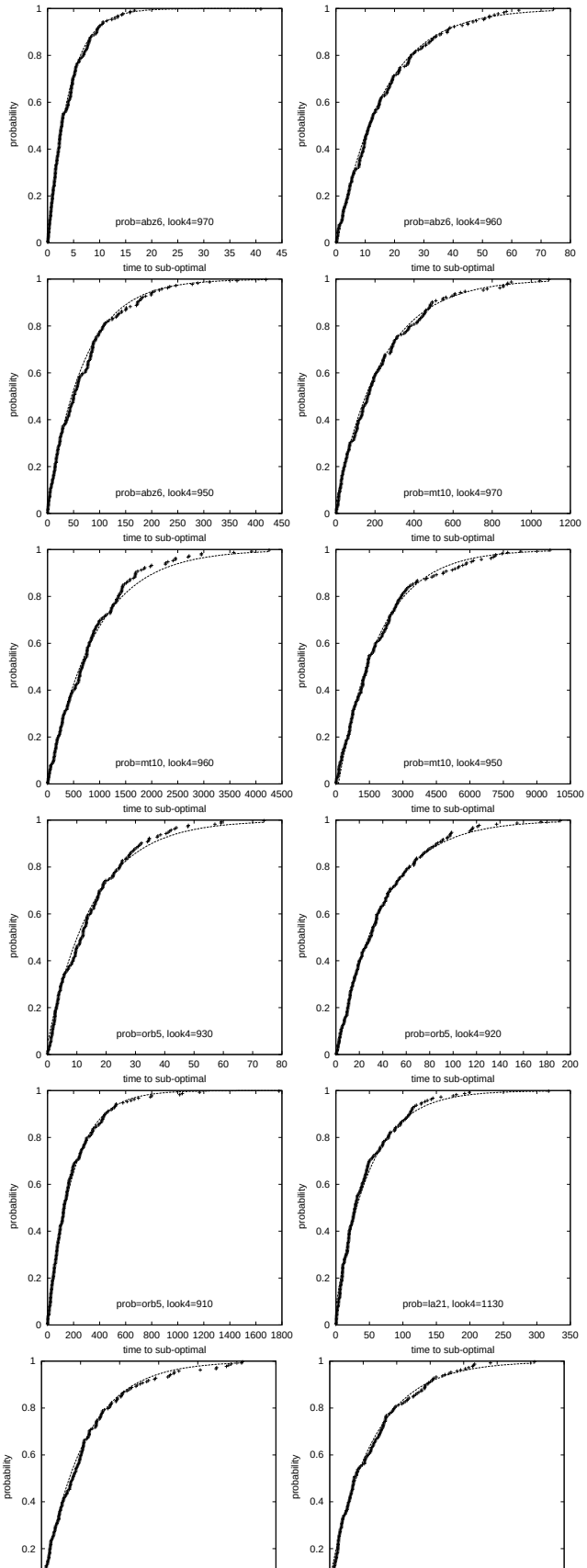
9.6 CONCLUDING REMARKS

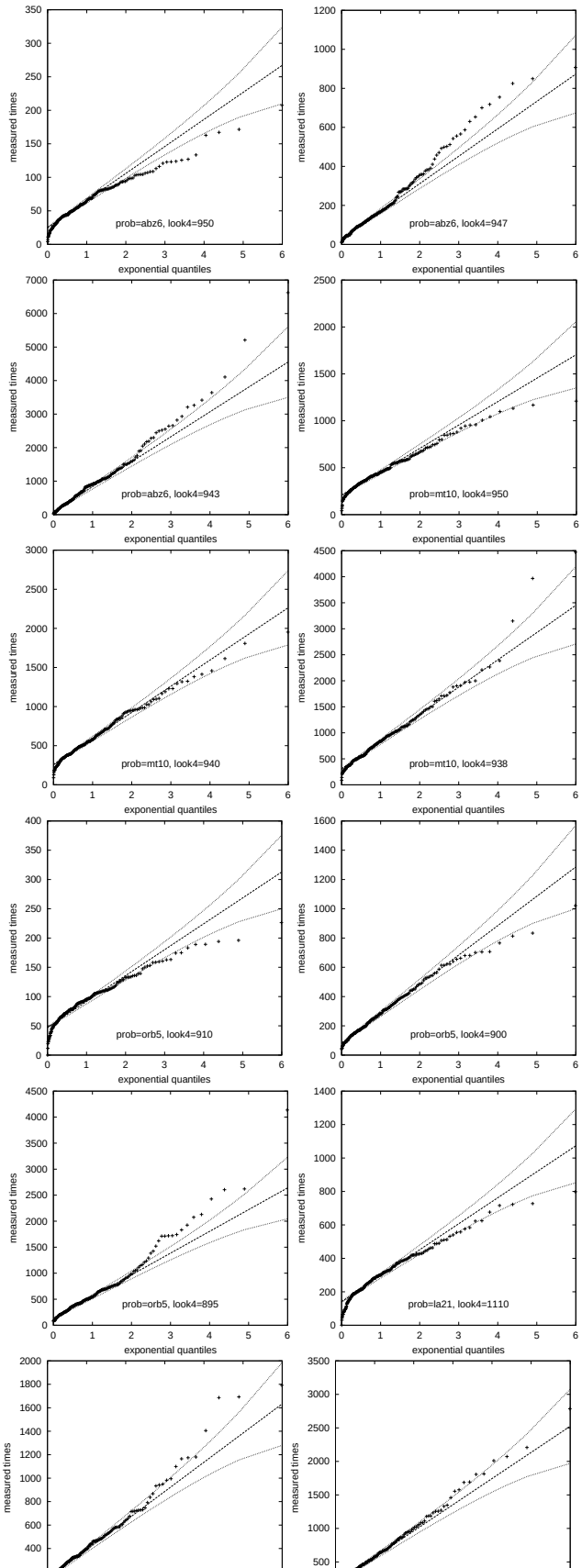
We describe a new algorithm for finding approximate solutions to the job shop scheduling problem. This GRASP uses some of the ideas proposed in the GRASP of Binato et al. (2001). That GRASP applies an intensification strategy during the construction phase that uses information obtained from "good" solutions to implement a memory-based procedure to influence the construction phase. In the hybrid GRASP proposed in this paper, the intensification phase is moved to the end of each GRASP iteration and is done using path-relinking. Due to the high computational requirements of path-relinking, only solutions accepted by a quality criteria undergo this procedure. Furthermore, the new GRASP alternates between two semi-greedy algorithms to construct solutions, which produces a higher variety of initial solutions for the local search. The algorithm was evaluated on 66 standard test problems and was shown to produce optimal or near-optimal solutions on all instances.

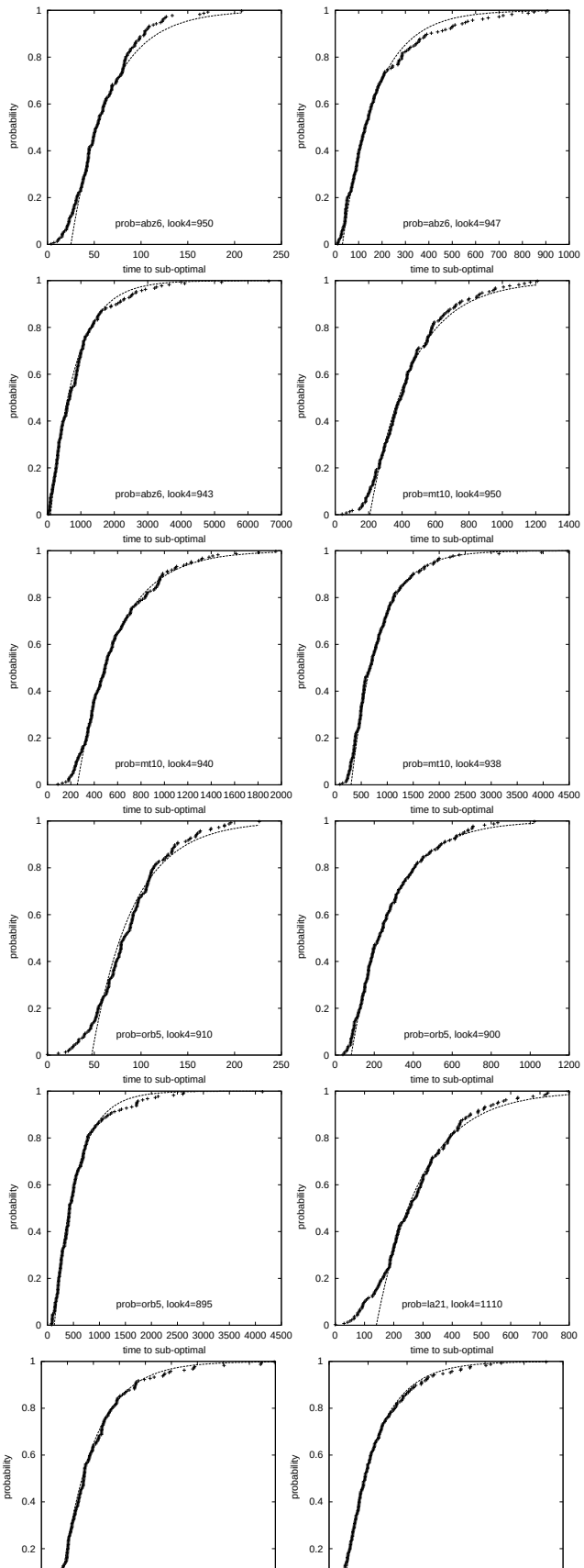
We observe that the hybrid GRASP with path-relinking obtains a solution of a given quality faster than the pure GRASP. Therefore, the increase in the computational time of each GRASP iteration due to the computation of path-relinking is compensated by an increase in the method's robustness. We also verify that the intensification applied after each GRASP iteration using path-relinking outperforms the intensification strategy used in Binato et al., which is applied during the construction phase.

We verify that the time to target sub-optimal solution of the proposed GRASPs fit well a two-parameter exponential distribution. Two parallelization strategies were proposed for the GRASP with path-relinking: an independent and a cooperative. The independent parallel strategy, as expected, shows a sub-linear speedup. The cooperative approach shows an approximate linear speedup for all instances tested, thus attesting that the extra time spent in communication among processes is compensated by an increase in solution quality.


measured times
exponential quantiles
prob=abz6, look4=970
prob=abz6, look4=960
prob=abz6, look4=950
prob=mt10, look4=970
prob=mt10, look4=960
prob=mt10, look4=950
prob=orb5, look4=930
prob=orb5, look4=920
prob=orb5, look4=910
prob=la21, look4=1130


probability
time to sub-optimal
prob=abz6, look4=970
prob=abz6, look4=960
prob=abz6, look4=950
prob=mt10, look4=970
prob=mt10, look4=960
prob=mt10, look4=950
prob=orb5, look4=930
prob=orb5, look4=920
prob=orb5, look4=910
prob=la21, look4=1130


prob=abz6, look4=950
prob=abz6, look4=947
prob=abz6, look4=943
prob=mt10, look4=950
prob=mt10, look4=940
prob=mt10, look4=938
prob=orb5, look4=910
prob=orb5, look4=900
prob=orb5, look4=895
prob=la21, look4=1110
measured times
exponential quantiles


probability
time to sub-optimal
prob=abz6, look4=950
prob=abz6, look4=947
prob=abz6, look4=943
prob=mt10, look4=950
prob=mt10, look4=940
prob=mt10, look4=938
prob=orb5, look4=910
prob=orb5, look4=900
prob=orb5, look4=895
prob=la21, look4=1110

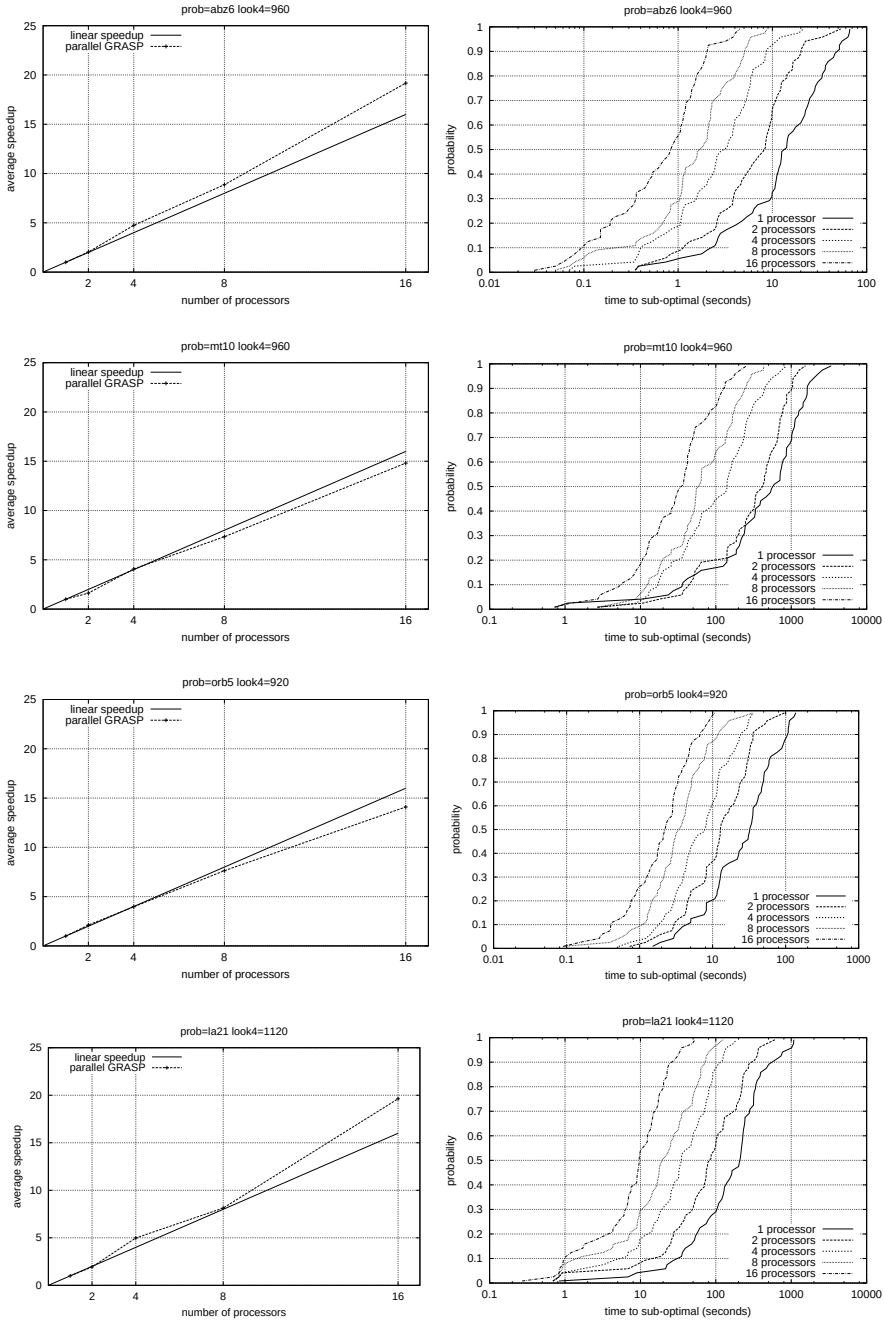


Figure 9.10 Speedup and empirical distributions for parallel implementation of GRASP: problems abz6, mt10, orb5 and la21.

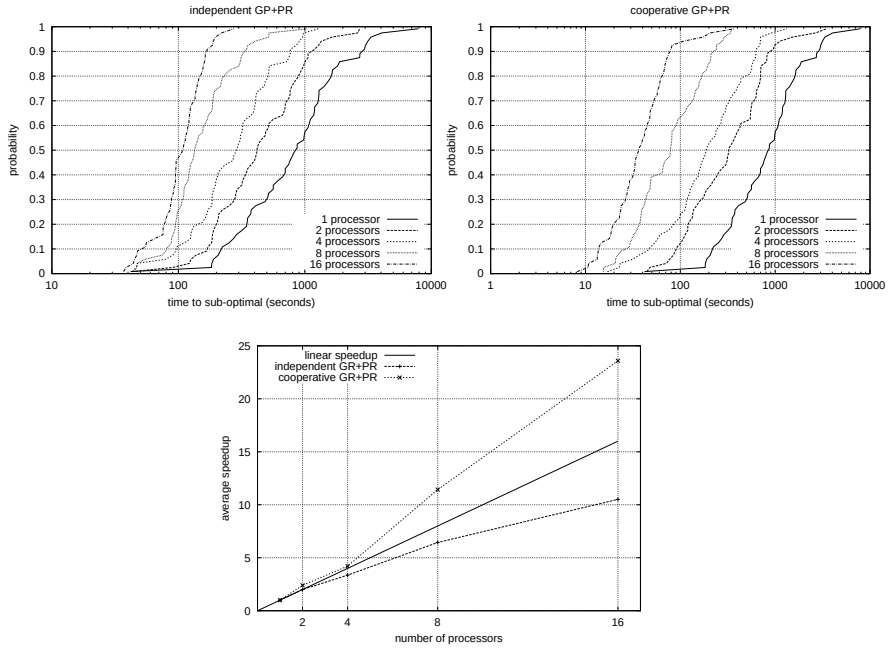


Figure 9.11 Speedup and empirical distributions for parallel implementations of GP+PR: problem abz6 with target value 943.

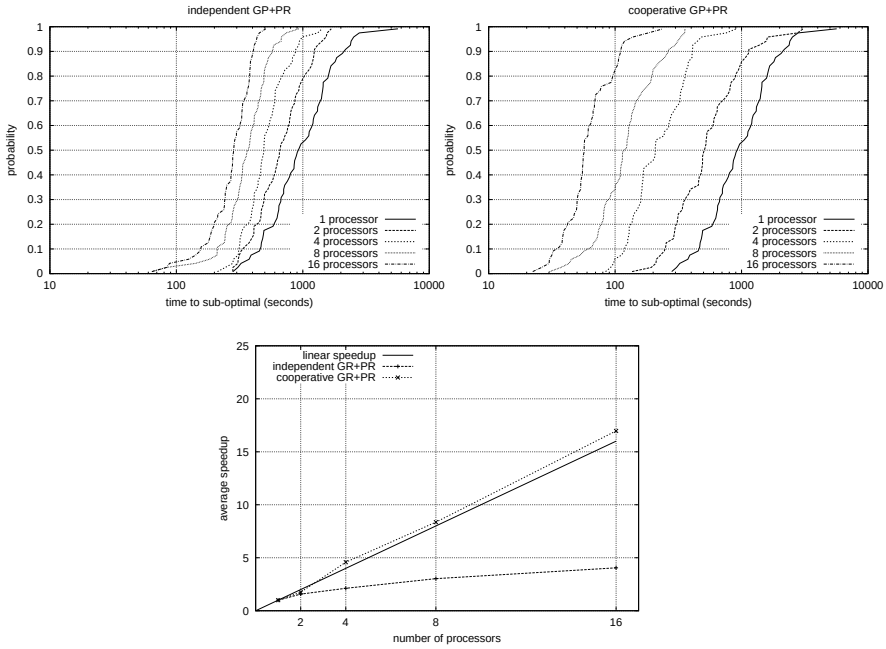


Figure 9.12 Speedup and empirical distributions for parallel implementations of GP+PR: problem mt10 with target value 938.

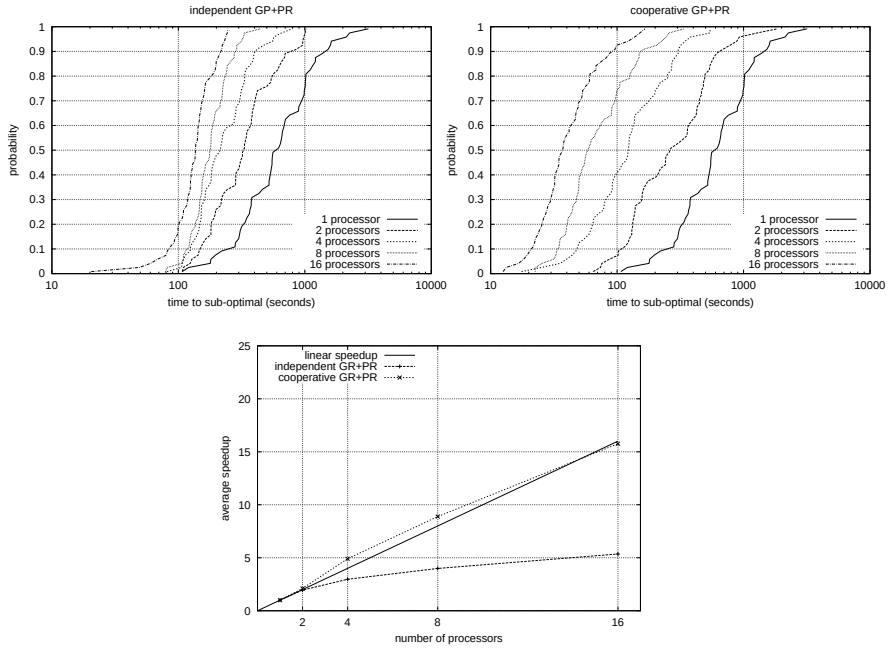


Figure 9.13 Speedup and empirical distributions for parallel implementations of GP+PR: problem orb5 with target value 895.

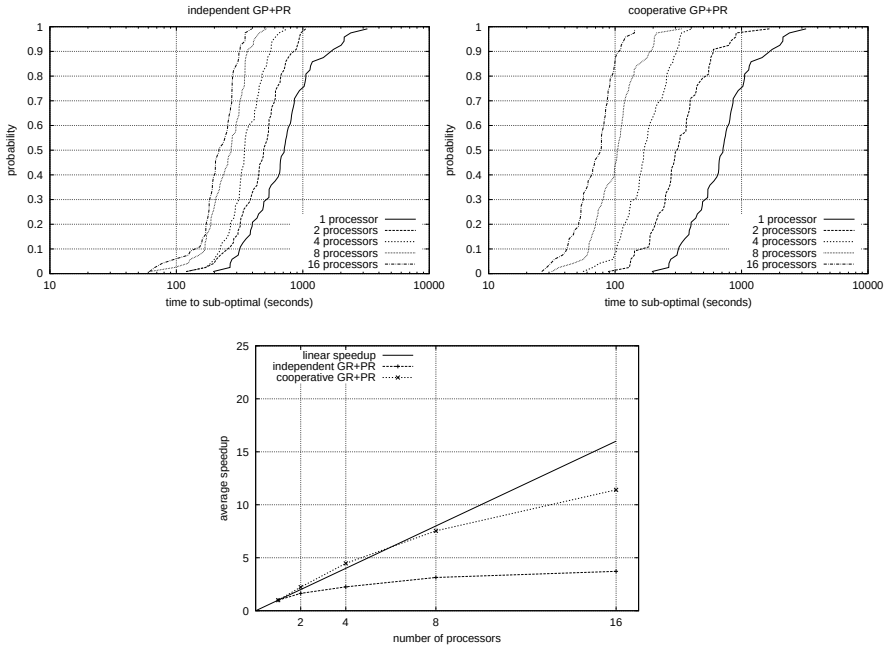


Figure 9.14 Speedup and empirical distributions for parallel implementations of `GP+PR`: problem `la21` with target value 1100.

```
procedure NON-COLLAB_GRASP_PR(M, J, O, p, seed, look4, maxitr, maxpool, freq)
1   my_rank = GET_RANK(); nprocs = GET_NUM_PROCS();
2   for i = 1, ..., (maxitr/nprocs) * my_rank do
3       seed = rand(seed);
4   rof;
5   P = ∅; num_stop = 0;
6   for i = 1, ..., ∞ do
7       if  mod (i, 2) == 0 then
8           GREEDY_MASKESPAN(seed, S, M, p, |M|, |J|, Makespan);
9       else
10          GREEDY_TIME_REMAINING(seed, S, M, p, |M|, |J|, Makespan);
11      fi;
12      LOCAL(S, M, p, |M|, |J|, Makespan);
13      if |P| == maxpool then
14          accepted = VERIFY_QUALITY(S, i);
15          if accepted then
16              for T ∈ P' ⊆ P do
17                  S_gmin = PATH_RELINKING(M, J, O, p, Makespan, S, T);
18                  UPDATE_POOL(S_gmin, c_gmin, P);
19                  S_gmin = PATH_RELINKING(M, J, O, p, Makespan, T, S);
20                  UPDATE_POOL(S_gmin, c_gmin, P);
21              rof;
22          fi;
23      else P = P ∪ {S} fi;
24      if  mod (i, ifreq) == 0 then INTENSIFY(P) fi;
25      S_best = POOLMIN(P);
26      if MAKESPAN(S_best) ≤ look4 then SEND_ALL(look4_stop) fi;
27      if i == maxitr then
28          num_stop = num_stop + 1;
29          SEND_ALL(maxitr_stop);
30      fi;
31      received = VERIFY_RECEIVING(flag);
32      if received then
33          if flag == look4_stop then break;
34          else if flag == maxitr_stop then num_stop = num_stop + 1 fi;
35      fi;
36      if num_stop == nprocs then break fi;
37  rof;
38  POSTOPT(POOL);
39  S_GlobalBest = GET_GLOBAL_BEST(S_best);
40  return (S_GlobalBest);
end NON-COLLAB_GRASP_PR;
```

Figure 9.15 Pseudo-code for the non-collaborative parallel GRASP with path-relinking.

```
procedure COLLAB_GRASP_PR(M, J, O, p, seed, look4, maxitr, maxpool, freq)
1   my_rank = GET_RANK(); nprocs = GET_NUM_PROCS();
2   for i = 1, ... , (maxitr/nprocs) * my_rank do
3       seed = rand(seed);
4   rof;
5   P = ∅; num_stop = 0;
6   for i = 1, ... , ∞ do
7       if  mod (i, 2) == 0 then
8           GREEDY_MASKESPAN(seed, S, M, p, |M|, |J|, Makespan);
9       else
10          GREEDY_TIME_REMAINING(seed, S, M, p, |M|, |J|, Makespan);
11      fi;
12      LOCAL(S, M, p, |M|, |J|, Makespan);
13      if |P| == maxpool then
14          accepted = VERIFY_QUALITY(S, i);
15          if accepted then
16              for T ∈ P' ⊆ P do
17                  RECEIVE_SOLUTIONS(P);
18                  S_gmin = PATH_RELINKING(M, J, O, p, Makespan, S, T);
19                  updated = UPDATE_POOL(S_gmin, c_gmin, P);
20                  if (updated) then INSERT_SEND_BUFFER(S_gmin, c_gmin, buffer) fi;
21                  RECEIVE_SOLUTIONS(P);
22                  S_gmin = PATH_RELINKING(M, J, O, p, Makespan, T, S);
23                  updated = UPDATE_POOL(S_gmin, c_gmin, P);
24                  if (updated) then INSERT_SEND_BUFFER(S_gmin, c_gmin, buffer) fi;
25              rof;
26              SEND_SOLUTIONS(buffer);
27          fi;
28      else P = P ∪ {S} fi;
29      if  mod (i, ifreq) == 0 then INTENSIFY(P) fi;
30      S_best = POOLMIN(P);
31      if MAKESPAN(S_best) ≤ look4 then SEND_ALL(look4_stop) fi;
32      if i == maxitr then
33          num_stop = num_stop + 1;
34          SEND_ALL(maxitr_stop)
35      fi;
36      received = VERIFY_RECEIVING(flag);
37      if received then
38          if flag == look4_stop then break;
39          else if flag == maxitr_stop then num_stop = num_stop + 1 fi;
40      fi;
41      if num_stop == nprocs then break fi;
42  rof;
43  POSTOPT(POOL);
44  S_GlobalBest = GET_GLOBAL_BEST(S_best);
45  return (S_GlobalBest);
end COLLAB_GRASP_PR;
```

Figure 9.16 Pseudo-code for collaborative parallel GRASP with path-relinking.

Bibliography

J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401, 1988.

R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with path-relinking for the three-index assignment problem. Technical report, AT&T Labs-Research, 2000a.

R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. Technical report, AT&T Labs Research, Florham Park, NJ 07733, 2000b.

D. Applegate and W.Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3:149–156, 1991.

J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

S. Binato, W.J. Hery, D. Loewenstern, and M.G.C. Resende. A GRASP for job shop scheduling. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys on metaheuristics*, pages 59–79. Kluwer Academic Publishers, 2001.

P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:105–127, 1994.

S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38:50–58, 2001.

J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35:164–176, 1989.

J. Carlier and E. Pinson. A practical use of Jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, 26:269–287, 1990.

J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. *Graphical Methods for Data Analysis*. Chapman & Hall, 1983.

L. Davis. Job shop scheduling with genetic algorithms. In *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, pages 136–140. Morgan Kaufmann, 1985.

T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys on metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2001.

H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice Hall, Englewood Cliffs, NJ, 1963.

C. Fleurent and F. Glover. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing*, 11: 198–204, 1999.

S. French. *Sequencing and scheduling:An introduction to the mathematics of the job-shop*. Horwood, 1982.

B. Giffler and G. L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1960.

F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

F. Glover. Multi-start and strategic oscillation methods – Principles to exploit adaptive memory. In M. Laguna and J.L. Gonzáles-Velarde, editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pages 1–24. Kluwer, 2000.

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

A. S. Jain and S. Meeran. A state-of-the-art review of job-shop scheduling techniques. Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, 1998.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS J. on Computing*, 11:44–52, 1999.

J. K. Lenstra and A. H. G. Rinnooy Kan. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.

H.R. Lourenço. Local optimization and the job-shop scheduling problem. *European Journal of Operational Research*, 83:347–364, 1995.

H.R. Lourenço and M. Zwijnenburg. Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory and Apllications*, pages 219–236. Kluwer Academic Publishers, 1996.

E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42:797–813, 1996.

E. Pinson. The job shop scheduling problem: A concise survey and some recent developments. In P. Chrétienne, E.G. Coffman Jr., J.K. Lenstra, and Z. Liu, editors, *Scheduling theory and its application*, pages 277–293. John Wiley and Sons, 1995.

M.G.C. Resende and C.C. Ribeiro. A grasp with path-relinking for permanent virtual circuit routing. Technical report, AT&T Labs Research, 2001a.

M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. Technical report, AT&T Labs Research, Florham Park, NJ, 2001b. To appear in *State-of-the-Art Handbook in Metaheuristics*, F. Glover and G. Kochenberger, eds., Kluwer Academic Publishers.

C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. Technical report, Computer Science Department, Catholic University of Rio de Janeiro, 2001.

B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives, 1964.

M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI – The complete reference, Volume 1 – The MPI Core*. The MIT Press, 1998.

E. D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6:108–117, 1994.

R. J. M. Vaessens, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by local search. *INFORMS Journal on Computing*, 8:302–317, 1996.

P. J. M. Van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.

M.G.A. Verhoeven and E.H.L. Aarts. Parallel local search. *J. of Heuristics*, 1:43–66, 1995.

III GRASP implementations for locations problems

10 A FAST SWAP-BASED LOCAL SEARCH PROCEDURE FOR LOCATION PROBLEMS

Mauricio G. C. Resende[1] and Renato F. Werneck[2]

[1]Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA.
mgcr@research.att.com

[2]Department of Computer Science
Princeton University
Princeton, NJ 08544 USA
rwerneck@cs.princeton.edu

Abstract: We present a new implementation of a widely used swap-based local search procedure for the p-median problem, proposed in 1968 by Teitz and Bart. Our method produces the same output as the best alternatives described in the literature and, even though its worst-case complexity is similar, it can be significantly faster in practice: speedups of up to three orders of magnitude were observed. We also show that our method can be easily adapted to handle the facility location problem and to implement related procedures, such as path-relinking and tabu search.
Keywords: Local search, location theory, p-median, facility location.

10.1 INTRODUCTION

The p-median problem is defined as follows. Given a set F of m facilities, a set U of n users (or customers), a distance function $d : U \times F \rightarrow \mathcal{R}_+$, and an integer $p \leq m$, determine which p facilities to open so as to minimize the sum of the distances from each user to the closest open facility. In other words, given p, we want to minimize the cost of serving all customers.

Since this problem is NP-hard (Kariv and Hakimi, 1979), a polynomial-time algorithm to solve it exactly is unlikely to exist. The most effective algorithms pro-

posed in the literature (Avella et al., 2003a; Beasley, 1985; Briant and Naddef, 2004; Cornuéjols et al., 1977; Galvão, 1980; Rosing et al., 1979; Senne et al., 2005) use branch-and-bound, with lower bounds obtained from some linear programming relaxation of the problem. In the worst case, all these methods are exponential, but they can be quite fast in practice (the recent algorithm by Avella et al. (2003a) is particularly effective). Also, similar techniques can be made to work as heuristics only, producing close-to-optimal solutions in reasonable time (Avella et al., 2003b; du Merle et al., 1999; Senne and Lorena, 2000; 2002).

There are also simpler heuristics that use no duality (linear programming) information at all. The most natural options are constructive heuristics, methods that build solutions from scratch, usually in a greedy fashion (Cornuéjols et al., 1977; Kuehn and Hamburger, 1963; Whitaker, 1983). A step further is to use a *local search procedure*, which takes an existing solution as input and tries to improve it (Goodchild and Noronha, 1983; Hodgson, 1978; Maranzana, 1964; Rosing, 1997; Taillard, 2003; Teitz and Bart, 1968). It does so in an iterative fashion, examining *neighboring solutions*, those that differ from the original one by a small (problem- and algorithm-specific) modification. Finally, there are metaheuristics, procedures that aim at exploring a large portion of the search space in an organized fashion to obtain close-to-optimal solutions (possibly using constructive algorithms and local search as subroutines). Recent examples in the literature include variable neighborhood search (Hansen and Mladenović, 1997), variable neighborhood decomposition search (Hansen et al., 2001), tabu search (Rolland et al., 1996; Voß, 1996), heuristic concentration (Rosing and ReVelle, 1997), scatter search (García-López et al., 2003), and a GRASP-based hybrid algorithm (Resende and Werneck, 2004).

This study concerns the local search proposed by Teitz and Bart (1968), based on swapping facilities. In each iteration, the algorithm looks for a pair of facilities (one to be inserted into the current solution, another to be removed) that would lead to an improved solution if swapped. If such a pair exists, the swap is made and the procedure is repeated.

Arya et al. (2001) have shown that, in a metric setting, this algorithm always finds solutions that are within a factor of at most 5 from the optimum. However, for practical, non-pathological instances the gap is usually much smaller, just a few percentage points (Rosing, 1997; Whitaker, 1983). This has made the algorithm very popular among practioners, often appearing as a key subroutine of more elaborate metaheuristics (García-López et al., 2003; Hansen and Mladenović, 1997; Resende and Werneck, 2004; Rolland et al., 1996; Rosing and ReVelle, 1997; Voß, 1996).

Our concern in this paper is not solution quality—the reader is referred to Rosing (1997) and Whitaker (1983) for insights on that matter. Our goal is to obtain the same solutions Teitz and Bart would, only in less time. We present an implementation that is significantly (often asymptotically) faster in practice than previously known alternatives.

The paper is organized as follows. In Section 10.2, we give a precise description of the local search procedure and a trivial implementation. In Section 10.3, we describe the best alternative implementation described in the literature, proposed by Whitaker (1983). Our own implementation is described in Section 10.4. We show how it can

be adapted to handle the facility location problem and to handle related operations (such as path-relinking and tabu search) in Section 10.5. Experimental evidence to the efficiency of our method is presented in Section 10.6. Final remarks are made in Section 10.7.

Notation and assumptions.. Before proceeding to the study of the algorithms themselves, let us establish some notation. As already mentioned, F is the set of potential facilities and U the set of users that must be served. The basic parameters of the problem are $n = |U|$, $m = |F|$, and p, the number of facilities to open. Although $1 \leq p \leq m$ by definition, we will ignore trivial cases and assume that $1 < p < m$ and that $p < n$ (if $p \geq n$, we just open the facility that is closest to each user). We assume nothing about the relationship between n and m.

We use u to denote a generic user, and f a generic facility. The cost of serving u with f is $d(u, f)$, the *distance* between them, which is always nonnegative. (We do not make any other assumption about the distance function; in particular, we do not assume that the triangle inequality is valid.) A *solution* S is any subset of F with p elements, and represents the set of open facilities. Every user u is assigned to the closest facility $f \in S$ (the one that minimizes $d(u, f)$). This facility will be denoted by $\phi_1(u)$. Our algorithm often needs to access the second closest facility to u in S as well; it will be denoted by $\phi_2(u)$. To simplify notation, we will abbreviate $d(u, \phi_1(u))$ as $d_1(u)$, and $d(u, \phi_2(u))$ as $d_2(u)$.[1] We often deal specifically with a facility that is a candidate for insertion; it will be referred to as f_i (by definition $f_i \notin S$); similarly, a candidate for removal will be denoted by f_r ($f_r \in S$, also by definition).

Throughout this paper, we assume the *distance oracle* model, in which the distance between any customer and any facility can be determined in $O(1)$ time. In this model, all values of ϕ_1 and ϕ_2 for a given solution S can be straighforwardly computed in $O(pn)$ total time: for each of the n customers, we explicitly find the distances to the p open facilities and pick the smallest. Problems defined by a distance matrix clearly fall into the distance oracle model, but an explicit matrix is not always necessary. If users and facilities are points on the plane, for example, distances can also be computed in constant time. There are cases, however, in which that does not happen, such as when the input is given as a sparse graph, with distances determined by shortest paths. In such situations, one must precompute the corresponding distance matrix in order to apply our method with the same worst-case running time.

10.2 THE SWAP-BASED LOCAL SEARCH

Introduced by Teitz and Bart (1968), the standard local search procedure for the p-median problem is based on swapping facilities. For each facility $f_i \notin S$ (the current solution), the procedure determines which facility $f_r \in S$ (if any) would improve the solution the most if f_i and f_r were interchanged (i.e., if f_i were inserted and f_r removed from the solution). If any such "improving" swap exists, the best one is performed, and

[1] More accurate representations of $\phi_1(u)$, $\phi_2(u)$, $d_1(u)$, and $d_2(u)$ would be $\phi_1^S(u)$, $\phi_2^S(u)$, $d_1^S(u)$, and $d_2^S(u)$, respectively, since each value is a function of S as well. Since the solution will be clear from context, we prefer the simpler representation in the interest of readability.

the procedure is repeated from the new solution. Otherwise we stop, having reached a *local minimum* (or *local optimum*). Arya et al. (2001) have recently proven that this procedure is guaranteed to produce a solution whose value is at most 5 times the optimum in the metric setting (i.e., when the triangle inequality holds). On non-pathological instances (those more likely to appear in practice), empirical evidence shows that the algorithm is often within a few percentage points of optimality (and often does find the optimal solution), being especially successful when both p and n are small (Rosing, 1997).

Our main concern is not solution quality, but the time it takes to run each iteration of the algorithm. Given a solution S, we want to find an improving neighbor S' (if it exists) as fast as possible.

A straighforward implementation takes $O(pmn)$ time per iteration. Start by determining the closest and second closest open facilities for each user; this takes $O(pn)$ time. Then, for each candidate pair (f_i, f_r), compute the profit that would result from replacing f_r with f_i. To do that, one can reason about each user u independently. If the facility that currently serves u is not f_r (the facility to be removed), the user will switch to f_i only if this facility is closer, otherwise it will remain where it is. If u is currently assigned to f_r, the user will have to be reassigned, either to $\phi_2(u)$ (the second closest facility) or to f_i (the facility to be inserted), whichever is closest. The net effect is summarized by following expression:

$$profit(f_i, f_r) = \sum_{u:\phi_1(u)\neq f_r} \max\{0, [d_1(u) - d(u, f_i)]\} - \sum_{u:\phi_1(u)=f_r} [\min\{d_2(u), d(u, f_i)\} - d_1(u)].$$

The first summation accounts for users that are not currently assigned to f_r (these can only gain from the swap), and the second for users that are (they can gain or lose something with the swap). In the distance oracle model, the entire expression can be computed in $O(n)$ time for each candidate pair of facilites. There are p candidates for removal and $m - p$ for insertion, so the total number of moves to consider is $p(m - p) = O(pm)$. Each iteration therefore takes $O(pmn)$ time.

Several papers in the literature use this basic implementation, and others avoid using the swap-based local search altogether mentioning its intolerable running time (Rolland et al., 1996; Rosing and ReVelle, 1997; Voß, 1996). These methods would greatly benefit from asymptotically faster implementations, such as Whitaker's or ours.

10.3 WHITAKER'S IMPLEMENTATION

Whitaker (1983) describes the so-called *fast interchange* heuristic, an efficient implementation of the local search procedure defined above. Even though it was published in 1983, Whitaker's implementation was not widely used until 1997, when Hansen and Mladenović (1997) applied it as a subroutine of a Variable Neighborhood Search (VNS) procedure. A minor difference between the implementations is that Whitaker prefers a *first improvement* strategy (a swap is made as soon as a profitable one is found), while Hansen and Mladenović prefer *best improvement* (all swaps are evaluated and the most profitable executed). In our analysis, we assume best improvement is used, even in references to "Whitaker's algorithm."

```
function findOut (S, f_i, φ_1, φ_2)
1    gain ← 0; /* gain resulting from the addition of f_i */
2    forall (f ∈ S) do netloss(f) ← 0; /* loss resulting from removal of f */
3    forall (u ∈ U) do
4        if (d(u, f_i) ≤ d_1(u)) then /* gain if f_i is close enough to u */
5            gain +← [d_1(u) − d(u, f_i)];
6        else /* loss if facility that is closest to u is removed */
7            netloss(φ_1(u)) +← min{d(u,f_i), d_2(u)} − d_1(u);
8        endif
9    endforall
10   f_r ← argmin_{f∈S}{netloss(f)};
11   profit ← gain − netloss(f_r);
12   return (f_r, profit);
end findOut
```

Figure 10.1 Function to determine, given a candidate for insertion (f_i), the best candidate for removal (f_r). Adapted from Hansen and Mladenović (1997).

The key aspect of this implementation is its ability to find in $\Theta(n)$ time the best possible candidate for removal, given a certain candidate for insertion. The pseudocode for the function that does that, adapted from Hansen and Mladenović (1997), is presented in Figure 10.1.[2] Function `findOut` takes as input a candidate for insertion (f_i) and returns f_r, the most profitable facility to be swapped out, as well as the profit itself (*profit*).

Given a certain candidate for insertion f_i, the function implicitly computes $profit(f_i, f_r)$ for all possible candidates f_r. What makes this procedure fast is the observation (due to Whitaker) that the profit can be decomposed into two components, which we call *gain* and *netloss*.

Component *gain* accounts for all users who would benefit from the insertion of f_i into the solution. Each is closer to f_i than to the facility it is currently assigned to. The difference between the distances is the amount by which the cost of serving that particular user will be reduced if f_i is inserted. Lines 4 and 5 of the pseudocode compute *gain*.

The second component, *netloss*, accounts for all other customers, those that would not benefit from the insertion of f_i into the solution. If the facility that is closest to u is removed, u would have to be reassigned either to $\phi_2(u)$ (its current second closest facility) or to f_i (the new facility), whichever is closest. In both cases, the cost of serving u will either increase or remain constant. Of course, this reassignment will only be necessary if $\phi_1(u)$ is the facility removed to make room for f_i. This explains why *netloss* is an array, not a scalar value: there is one value associated with each

[2] In the code, an expression of the form $a \stackrel{+}{\leftarrow} b$ means that the value of a is incremented by b units.

candidate for removal. All values are initially set to zero in line 2; line 7 adds the contributions of the relevant users.

Given this $O(n)$-time function, it is trivial to implement the swap-based local search procedure in $O(mn)$ time per iteration: simply call `findOut` once for each of the $m-p$ candidates for insertion and pick the most profitable one. If the best profit is positive, perform the swap, update the values of ϕ_1 and ϕ_2, and proceed to the next iteration. Updating ϕ_1 and ϕ_2 requires $O(pn)$ time in the worst case, but the procedure can be made faster in practice, as mentioned in (Whitaker, 1983). Since our implementation uses the same technique, its description is deferred to the next section (Subsection 10.4.3.1).

10.4 AN ALTERNATIVE IMPLEMENTATION

Our implementation has some similarity with Whitaker's, in the sense that both methods perform the same basic operations. However, the order in which they are performed is different, and in our case partial results are stored in auxiliary data structures. As we will see, with this approach we can use values computed in early iterations of the local search procedure to speed up later ones.

10.4.1 Additional Structures

Before we present our algorithm, let us analyze Whitaker's algorithm from a broader perspective. Its ultimate goal is to determine the pair (f_i, f_r) of facilities that maximizes $\mathit{profit}(f_i, f_r)$. To do so, it computes $\mathit{gain}(f_i)$ for every candidate for insertion, and $\mathit{netloss}(f_i, f_r)$ for every pair of candidates. (In the description in Section 10.3, *gain* is a scalar and *netloss* takes as input only the facility to be removed; however, both are computed inside a function that is called for each f_i, which accounts for the additional dimension.) Implicitly, what the algorithm does is to compute profits as

$$\mathit{profit}(f_i, f_r) = \mathit{gain}(f_i) - \mathit{netloss}(f_i, f_r).$$

Our algorithm defines $\mathit{gain}(f_i)$ precisely as in Whitaker's algorithm: it represents the total amount gained if f_i is added to S, regardless of which facility is removed:

$$\mathit{gain}(f_i) = \sum_{u \in U} \max\{0, d_1(u) - d(u, f_i)\}. \tag{10.1}$$

Our method differs from Whitaker's in the computation of *netloss*. While Whitaker's algorithm computes it explicitly, we do it in an indirect fashion. For every facility f_r in the solution, we define $\mathit{loss}(f_r)$ as the increase in solution value that results from the removal of f_r from the solution (assuming that no facility is inserted). This is the cost of transferring every customer assigned to f_r to its second closest facility:

$$\mathit{loss}(f_r) = \sum_{u:\phi_1(u)=f_r} [d_2(u) - d_1(u)]. \tag{10.2}$$

As defined, *gain* and *loss* are capable of determining the net effect of a single insertion or a single deletion, but not of a swap, which is nothing but an insertion and

a deletion that occur simultaneously. Whitaker's algorithm can handle swaps because it computes *netloss* instead of *loss*. To compute *netloss* from *loss*, we use yet another function, $extra(f_i, f_r)$, defined so that the following is true for all pairs (f_i, f_r):

$$netloss(f_i, f_r) = loss(f_r) - extra(f_i, f_r). \tag{10.3}$$

From the pseudocode in Figure 10.1, it is clear that $netloss(f_i, f_r)$ is actually defined as

$$netloss(f_i, f_r) = \sum_{\substack{u:[\phi_1(u)=f_r]\wedge \\ [d(u,f_i)>d_1(u)]}} [\min\{d(u, f_i), d_2(u)\} - d_1(u)]. \tag{10.4}$$

Substituting the values in Equations 10.2 and 10.4 into Equation 10.3, we obtain an expression for *extra*:

$$extra(f_i, f_r) = \sum_{u:\phi_1(u)=f_r} [d_2(u) - d_1(u)] - \sum_{\substack{u:[\phi_1(u)=f_r]\wedge \\ [d(u,f_i)>d_1(u)]}} [\min\{d(u, f_i), d_2(u)\} - d_1(u)].$$

It is possible to simplify this expression. First, consider a user u for which $\min\{d(u, f_i), d_2(u)\} = d_2(u)$. It has no net contribution to *extra*: whatever is added in the first summation is subtracted in the second. Therefore, we can write

$$extra(f_i, f_r) = \sum_{\substack{u:[\phi_1(u)=f_r]\wedge \\ [d(u,f_i)<d_2(u)]}} [d_2(u) - d_1(u)] - \sum_{\substack{u:[\phi_1(u)=f_r]\wedge \\ [d_1(u)<d(u,f_i)<d_2(u)]}} [d(u, f_i) - d_1(u)].$$

Note that the range of the first summation contains that of the second. We can join both into a single summation,

$$extra(f_i, f_r) = \sum_{\substack{u:[\phi_1(u)=f_r]\wedge \\ [d(u,f_i)<d_2(u)]}} [d_2(u) - d_1(u) - \max\{0, d(u, f_i) - d_1(u)\}],$$

which can be further simplified to

$$extra(f_i, f_r) = \sum_{\substack{u:[\phi_1(u)=f_r]\wedge \\ [d(u,f_i)<d_2(u)]}} [d_2(u) - \max\{d(u, f_i), d_1(u)\}]. \tag{10.5}$$

This is our final expression for *extra*. We derived it algebraically from simpler expressions, but it is possible to get it directly with a bit of case analysis. This alternative approach was used in an earlier version of our paper (Resende and Werneck, 2003).

Given the expressions of *gain*, *loss*, and *extra* (Equations 10.1, 10.2, and 10.5), we can find the profit associated with each move in a very simple manner:

$$profit(f_i, f_r) = gain(f_i) - loss(f_r) + extra(f_i, f_r). \tag{10.6}$$

The interesting aspect of this decomposition of *profit* is that the only term that depends on both the facility to be inserted and the one to be removed is *extra*. Moreover, this term is always nonnegative (see Equation 10.5). This will be relevant in the implementation of the local search itself, as the next section will make clear.

```
function updateStructures (S, u, loss, gain, extra, φ1, φ2)
1     f_r ← φ1(u);
2     loss(f_r) +← [d2(u) − d1(u)];
3     forall (f_i ∉ S) do
4         if (d(u, f_i) < d2(u)) then
5             gain(f_i) +← max{0, d1(u) − d(u, f_i)};
6             extra(f_i, f_r) +← [d2(u) − max{d(u, f_i), d1(u)}];
7         endif
8     endfor
end updateStructures
```

Figure 10.2 Pseudocode for updating arrays in the local search procedure

10.4.2 Local Search

Our implementation of the local search procedure assumes that all necessary values (*loss*, *gain*, and *extra*) are stored in appropriate data structures: one-dimensional vectors for *loss* and *gain*, and a two-dimensional matrix for *extra*.[3] Once these structures are computed, one can easily find the best swap in $O(pm)$ time: just use Equation 10.6 to determine the profit for each candidate pair of facilities and pick the minimum.

To compute *gain*, *loss*, and *extra*, we note that every entry in these structures is a summation over some subset of users (see Equations 10.1, 10.2, and 10.5). The contribution of each user can therefore be computed independently. Function `updateStructures`, shown in Figure 10.2, does exactly that. Given a user u and its two closest facilities in solution S (given by ϕ_1 and ϕ_2), it adds u's contribution to *loss*, *gain*, and *extra*. The total running time of the procedure is $O(m-p) = O(m)$, since it is essentially a loop through all the facilities that do not belong to the solution. Given this function, computing *gain*, *loss*, and *extra* from scratch is straightforward: first reset all entries in these structures, then call `updateStructures` once for each user. Together, these n calls perform precisely the summations defined in Equations 10.1, 10.2, and 10.5.

We now have all the elements necessary to build the local search procedure with $O(mn)$ operations. In $O(pn)$ time, compute $\phi_1(\cdot)$ and $\phi_2(\cdot)$ for all users. In $O(pm)$ time, reset *loss*, *gain*, and *extra*. With n calls to `updateStructures`, each taking in $O(m)$ time, determine their actual values. Finally, in $O(pm)$ time, find the best swap using Equation 10.6.

10.4.3 Acceleration

At first, our implementation seems to be merely a complicated alternative to Whitaker's; after all, both have the same worst-case complexity. Furthermore, our implementation has the clear disadvantage of requiring an $O(pm)$-sized matrix, whereas $\Theta(n+m)$

[3]Note that *gain* and *loss* could actually share the same m-sized vector, since they are defined for disjoint sets of facilities.

memory positions are enough for Whitaker's. The additional memory, however, allows for significant accelerations, as this section will show.

When a facility f_r is replaced by a new facility f_i, certain entries in *gain*, *loss*, *extra*, ϕ_1, and ϕ_2 become inaccurate. The straighforward way to update them for the next local search iteration is to recompute ϕ_1 and ϕ_2, reset the other arrays, and then call `updateStructures` again for all users.

A downside of this approach is that no information gathered in one iteration is used in subsequent ones. As a result, unnecessary, repeated computations are bound to occur. In fact, the actions performed by `updateStructures` depend only on u, $\phi_1(u)$, and $\phi_2(u)$; no value is read from other structures. If $\phi_1(u)$ and $\phi_2(u)$ do not change from one iteration to another, u's contribution to *gain*, *loss*, and *extra* will not change either. This means there is no need to call `updateStructures` again for u.

To deal with such cases, we keep track of *affected users*. A user u is *affected* if there is a change in either $\phi_1(u)$ or $\phi_2(u)$ (or both) after a swap is made. Sufficient conditions for u to be affected after a swap between f_i and f_r are:

1. either $\phi_1(u)$ or $\phi_2(u)$ is f_r, the facility removed; or
2. f_i (the facility inserted) is closer to u than the original $\phi_2(u)$ is.

Contributions to *loss*, *gain*, and *extra* need only be updated for affected users. If there happens to be few of them (which is often the case, as Section 10.6.2.1 shows) significant gains can be obtained.

Note, however, that updating the contributions of an affected user u requires more than a call to `updateStructures`. This function simply adds new contributions, so we must first subtract the old contributions made by u. To acomplish this, we use a function similar to `updateStructures`, with subtractions instead of additions.[4] This function (`undoUpdateStructures`) must be called for all affected users *before* ϕ_1 and ϕ_2 are recomputed.

Figure 10.3 contains the pseudocode for the entire local search procedure, already taking into account the observations just made. Apart from the functions already discussed, three other nontrivial ones appear in the code. Function `resetStructures`, sets all entries in *gain*, *loss*, and *extra* to zero. Function `findBestNeighbor` runs through these structures and finds the most profitable swap using Equation 10.6. It returns which facility to remove (f_r), the one to replace it (f_i), and the profit itself (*profit*). Finally, `updateClosest` updates ϕ_1 and ϕ_2, possibly using the fact that the facility recently opened was f_i and the one closed was f_r (Section 10.4.3.1 explains how this is done).

Restricting updates to affected users can result in significant speedups in the algorithm, as Section 10.6.2.1 shows. There are, however, other accelerations to exploit. The pseudocode reveals that all operations in the main loop run in linear time, with three exceptions:

- updating closeness information (calls to `updateClosest`);

[4]This function is identical to the one shown in Figure 10.2, with all occurrences of $\stackrel{+}{\leftarrow}$ replaced with $\stackrel{-}{\leftarrow}$: instead of incrementing values, we decrement them.

```
procedure localSearch (S, φ1, φ2)
1   A ← U; /* A is the set of affected users */
2   resetStructures (gain, loss, extra);
3   while (TRUE) do
4       forall (u ∈ A) do updateStructures (S, u, gain, loss, extra, φ1, φ2);
5       (fr, fi, profit) ← findBestNeighbor (gain, loss, extra);
6       if (profit ≤ 0) then break; /* no improvement, we are done */
7       A ← ∅;
8       forall (u ∈ U) do /* find out which users will be affected */
9           if ((φ1(u) = fr) or (φ2(u) = fr) or (d(u, fi) < d(u, φ2(u)))) then
10              A ← A ∪ {u}
11          endif
12      endforall;
13      forall (u ∈ A) do undoUpdateStructures (S, u, gain, loss, extra, φ1, φ2);
14      insert(S, fi);
15      remove(S, fr);
16      updateClosest(S, fi, fr, φ1, φ2);
17  endwhile
end localSearch
```

Figure 10.3 Pseudocode for the local search procedure

- finding the best swap to be made (calls to `findBestNeighbor`);
- updating the auxiliary data structures (calls to `updateStructures` and `undoUpdateStructures`).

These are the potential bottlenecks of the algorithm, since they all run in quadratic time in the worst case. The next three subsections analyze how each of them can be dealt with.

10.4.3.1 Closeness. Updating closeness information, in our experience, has proven to be a relatively cheap operation. Deciding whether the newly inserted facility f_i becomes either the closest or the second closest facility to each user is trivial and can be done in $O(n)$ total time. A more costly operation is updating closeness information for customers who had f_r (the facility removed) as either the closest or the second closest element. With a straighforward implementation, updating each such affected user takes $O(p)$ time. Since there are usually few of them, the total time spent tends to be a small fraction of the entire local search procedure.

The whole update procedure could actually be performed in $O(n \log p)$ worst-case time. It suffices to keep, for each user u, the set of open facilities in a heap with priorities given by their distances to u. Since this solution requires $O(np)$ additional memory positions and is not significantly faster, we opted for using the straighforward implementation in our code.

It is also important to mention that finding the set of closest and second closest elements from scratch is itself a cheap operation in some settings, even in the worst case. For example, when distances between customers and facilities are given by shortest

paths on an underlying graph, this can be accomplished in $\tilde{O}(|E|)$ time (Thorup, 2001), where $|E|$ is the number of edges in the graph.[5]

In practice, the generic approach above seems to be good enough. Section 10.6.2.5 shows that there is not much to gain from accelerating this part of the algorithm; together, other procedures already dominate the running time of the local search. We therefore do not use specialized routines in this paper; we always assume we are dealing with arbitrary distance matrices.

10.4.3.2 Best Neighbor. Given a solution, the straighforward way to find the most profitable swap is to compute $\mathit{profit}(f_i, f_r)$ (as defined in Equation 10.6) for all candidate pairs of facilities and pick the best. Since each *profit* computation takes constant time and there are $p(m-p)$ potential swaps, the entire procedure requires $\Theta(pm)$ operations. In practice, however, the best move can be found in less time.

It is convenient to think of $\mathit{extra}(f_i, f_r)$ as a measure of the interaction between the neighborhoods of f_r and f_i. After all, Equation 10.5 shows that only users that have f_r as their current closest facility and are also close to f_i (i.e., have f_i closer than the second closest open facility) contribute to $\mathit{extra}(f_i, f_r)$. In particular, if there are no users in this situation, $\mathit{extra}(f_i, f_r)$ will be zero. Section 10.6.2.2 shows that this occurs rather frequently in practice, especially when p is large (and hence the average number of users assigned to each f_r is small).

Therefore, instead of storing *extra* as a full matrix, one may consider representing only nonzero elements explicitly: each row becomes a linked list sorted by column number. A drawback of this sparse representation is the impossibility to make random accesses in $O(1)$ time. Fortunately, this is not necessary for our purposes. All three functions that access the matrix (`updateStructures`, `undoUpdateStructures`, and `bestNeighbor`) can be implemented so as to go through each row sequentially.

In particular, consider the implementation of `bestNeighbor`. First, it determines the facility f_i^* that maximizes $\mathit{gain}(f_i)$ and the facility f_r^* that minimizes $\mathit{loss}(f_r)$. Since all values in *extra* are nonnegative, the pair (f_i^*, f_r^*) is at least as profitable as any pair (f_i, f_r) for which $\mathit{extra}(f_i, f_r)$ is zero. Then, the procedure computes the exact profits (given by Equation 10.6) for all nonzero elements in *extra*.

The whole procedure takes $O(m + \lambda pm)$ time, where λ is the fraction of pairs whose *extra* value is nonzero. As already mentioned, this value tends to be smaller as p increases, thus making the algorithm not only faster, but also more memory-efficient (when compared to the "full matrix" representation).

10.4.3.3 Updates. As we have seen, keeping track of affected users can reduce the number of calls to `updateStructures`. We now study how to reduce the time spent in each of these calls.

Consider the pseudocode in Figure 10.2. Line 5 represents a loop through all $m-p$ facilities outside the solution, but line 6 shows that we can actually restrict ourselves to facilities that are closer to u than $\phi_2(u)$ is. This is often a small subset of the facilities, especially when p is large.

[5]The $\tilde{O}(\cdot)$ notation hides polylogarithmic terms.

This suggests a preprocessing step that builds, for each user u, a list of all facilities sorted by increasing distance to u. During the local search, whenever we need the set of facilities whose distance to u is less than $d_2(u)$, we just take the appropriate prefix of the precomputed list, potentially with much fewer than $m - p$ elements.

Building these lists takes $O(nm \log m)$ time, but it is done only once, not in every iteration of the local search procedure. This is true even if local search is applied several times within a metaheuristic (such as in Hansen and Mladenović (1997); Resende and Werneck (2003); Rosing and ReVelle (1997)): a single preprocessing step is enough.

A more serious drawback of this approach is memory usage. Keeping n lists of size m in memory requires $\Theta(mn)$ space, which may be prohibitive. An alternative is to keep only relatively small prefixes, not the full list. They would act as a cache: when $d_2(u)$ is small enough, we just take a prefix of the candidate list; when $d_2(u)$ is larger than the largest distance represented, we explicitly look at all possible neighbors (each in constant time).

In some circumstances, the "cached" version may be faster than the "full" version of the algorithm, since preprocessing is cheaper. After all, instead of creating sorted lists of size m, we create smaller ones of size k (for some $k < m$). Each list can be created in $O(m + k \log k)$ time: first we find the k smallest elements among all m in $O(m)$ time (Cormen et al., 2001), then we sort them in $O(k \log k)$ time. For small values of k, this is an asymptotic improvement over the $O(m \log m)$ time required (per list) in the "full" case.

10.4.3.4 The Reordering Problem. There is a slight incompatibility between the accelerations proposed in Sections 10.4.3.2 and 10.4.3.3. On the one hand, the sparse matrix data structure proposed in Section 10.4.3.2 guarantees efficient queries only when each row is accessed sequentially by column number (facility label). Section 10.4.3.3, on the other hand, assumes that facilites are accessed in nondecreasing order of *distance* from the user. Functions `updateStructures` and `undoUpdateStructures` use both data structures: they take a list of facilities sorted by distance, but must process them in nondecreasing order of label. We need to make these two operations compatible.

The simplest solution is to take the list of facilities sorted by distance and sort it again by label. If the list has size k, this takes $O(k \log k)$ time. In the worst case k is $O(m)$, so this introduces an extra $\log m$ factor in the complexity of the algorithm. In practice, however, k is rather small, and the overhead hardly noticeable. In fact, we used this approach in a preliminary version of our paper (Resende and Werneck, 2003).

Even so, one would like to do better. Recall that the original list is actually a prefix of the list of all facilities (sorted by distance). Even though the prefix varies in size, the underlying sorted list does not: it is a fixed permutation of facility labels. This means we need to solve the following generic problem:

> Let π be a fixed permutation of the labels $\{1, 2, \ldots, m\}$, and let π_k be the size-k prefix of π, for $1 \leq k \leq n$ ($\pi_n = \pi$, by definition). Given any k, sort π_k by label in $O(k)$ time. At most $O(m)$ preprocessing time is allowed.

To solve this, we use an algorithm that mimics insertion sort on a list, but takes advice from an "oracle" built during preprocessing. Assume we need to sort π_k, for some k. One way to do it is to take each element of π_k and insert it into a new list, ordered by label. With standard insertion sort, this would take $O(k^2)$ time. However, if we knew in advance where to insert each element, the procedure would take $O(k)$ time. The oracle will give us exactly that.

Let $\pi(i)$ be the i-th element of π. We define $pred(i)$ to be the *predecessor* of $\pi(i)$, the element after which $\pi(i)$ should be inserted during the algoritm above. The oracle will give us $pred(i)$ for every i.

The values of $pred(i)$ are set in the preprocessing step. Initially, it creates an auxiliary doubly-linked list L containing $0, 1, 2, \ldots, m$, in this order (element 0 will act as a sentinel). This can be trivially done in $O(m)$ time. Then, it removes elements from L one by one in *reverse order* with respect to π. In other words, the first element removed from L is $\pi(m)$, then $\pi(m-1)$, and so on, until $\pi(1)$ is removed; in the end, only 0 (the sentinel) will remain in L. Upon removing element $\pi(i)$ from L, the algorithm sets $pred(i)$ to be the predecessor of $\pi(i)$ (in L itself) at that particular moment. This procedure takes $O(m)$ time for each of the n lists.

Note that this procedure is in fact a simulation of insertion sort, but in reverse order. List L originally has all the elements of π_m; after one removal, we are left with π_{m-1}, and so on. At all times, L is sorted by label; if it has size k, it represents what the sequence looks like after the k-th element is inserted during insertion sort.

Given all the $pred(\cdot)$ values, sorting π_k is simple. We start with a list L' containing only a sentinel (0); it can be singly-linked, with forward pointers only. We then access the first i elements of π (following π's own order), inserting each element $\pi(i)$ into L' right after $pred(i)$. Eventually, L' will contain all the elements of $\pi(k)$ sorted by label, as desired. The running time is only $O(k)$.

10.5 GENERALIZATION

Section 10.4 presented our algorithm as a local search procedure for the p-median problem. In fact, with slight modifications, it can also be applied to the facility location problem. Moreover, the ideas suggested here are not limited to local search: they can also be used to accelerate other important routines, such as path-relinking and tabu search. This section details the adaptations that must be made in each case.

10.5.1 Facility Location

The input of the *facility location problem* consists of a set of users U, a set of potential facilities F, a distance function $d : U \times F \to \mathcal{R}_+$, and a *setup cost function* $c : F \to \mathcal{R}_+$. The first three parameters are the same as in the p-median problem. The difference is that here the number of facilities to open is not fixed; there is, instead, a cost associated with opening each facilty, the setup cost. The more facilities are opened, the greater the setup cost will be. The objective is to minimize the total cost of serving all customers, considering the sum of the setup and service cost (distances).

Any valid solution to the p-median problem is a valid solution to the facility location problem. To use the local search procedure suggested here for this problem,

we have to adjust the algorithm to compute the cost function correctly. As it is, the algorithm computes the service costs correctly, but assumes that the setup costs are zero. But including them is trivial: the service cost depends only on whether a facility is open or not; it does not depend on other facilities. Consider a facility f_i that is not in the solution; when evaluating whether it should be inserted or not, we must account for the fact that its setup cost will increase the solution value by $c(f_i)$. Similarly, simply closing a facility f_r that belongs to the solution saves us $c(f_r)$. To take these values into account, if suffices to initialize *gain* and *loss* with the symmetric of the corresponding setup costs, and not with zero as we do with the p-median problem. In other words, we initialize $gain(f_i)$ with $-c(f_i)$, and $loss(f_r)$ with $-c(f_r)$.

This is enough to implement a swap-based local search for the facility location problem. Note, however, that there is no reason to limit ourselves to swaps—we could allow individual insertions and deletions as well. This is not possible with the p-median problem because the number of facilities is fixed, but there is no such constraint in the facility location problem.

No major change to the algorithm is necessary to support individual insertions and deletions. As already mentioned, $gain(f_i)$ is exactly the amount that would be saved if facility f_i were inserted into the solution (with no corresponding removal). Similarly, $loss(f_r)$ represents how much would be lost if the facility were removed (with no corresponding insertion). Positive values of *gain* and negative values of *loss* indicate that the corresponding move is worth making. The greater the absolute value, the better, and we can find the maximum in $O(m)$ time. Furthermore, we can continue to compute the costs associated with swaps if we wish to. In every iteration of the local search, we could therefore choose the best move among all swaps, insertions, and deletions. So we essentially gain the ability to make insertions and deletions with barely any changes to the algorithm.

We observe that the idea of a swap-based local search for the facility location problem is, of course, not new; it was first suggested in the literature by Kuehn and Hamburger (1963).

10.5.2 Other Applications

It is possible to adapt the algorithm to perform other routines, not only local search. (In this discussion, we will always deal with the p-median problem itself, although the algorithms suggested here also apply to facility location with minor adaptations.)

Consider the path-relinking operation (Glover, 1996; Glover et al., 2000; Laguna and Martí, 1999; Resende and Ribeiro, 2005). It takes two solutions as inputs, S_1 and S_2, and gradually transforms the first (the *starting solution*) into the second (the *guiding solution*). It does so by swapping out facilities that are in $S_1 \setminus S_2$ and swapping in facilities from $S_2 \setminus S_1$. In each iteration of the algorithm, the best available swap is made. The goal of this procedure is to discover some promising solutions on the path from S_1 to S_2. The precise use of these solutions varies depending on the metaheuristic using this procedure.

This function is remarkably similar to the swap-based local search procedure. Both are based on the same kind of move (swaps), and both make the cheapest move on each round. There are two main differences:

1. *Candidate moves*: In path-relinking, only a subset of the facilities in the solution are candidates for removal, and only a subset of those outside the solution are candidates for insertion—and these subsets change (i.e., get smaller) over time, as the algorithm advances into the path.

2. *Stopping criterion*: Whereas the local search procedure stops as soon as a local minimum is found, non-improving moves are allowed in path-relinking: it continues until the guiding solution is reached.

As long as we take these differences into account, the implementation of the local search procedure can also handle path-relinking. We need to define two functions: one to return the appropriate set of candidates for insertion and deletion, another to check if the move chosen by `bestNeighbor` should be made or not (i.e., to determine if the stopping criterion was met). In Section 10.4, these functions were defined implicitly: the candidates for insertion are all facilities outside the solution, the candidates for deletion are those in the solution, and the stopping criterion consists of testing whether the profit associated with a move is positive. Defining them explicitly is trivial for both local search and path-relinking.

In fact, by redefining these two functions appropriately, we can implement other routines, such as a simple version of tabu search. At all times, we could have two lists: one for elements that are forbidden to be inserted into the solution, another for elements that cannot be removed. The candidate lists would contain the remaining facilities, and the stopping criterion could be any one used for tabu search (number of iterations, for instance).

10.6 EMPIRICAL ANALYSIS

This section has two main goals. One is to present some empirical data to back up some of the claims we have made to guide our search for a faster algorithm. The other goal is to demonstrate that the algorithms suggested here are indeed faster than previously existing implementations of the local search procedure for the p-median problem. To keep the analysis focused, we will not deal with the extensions proposed in Section 10.5.

10.6.1 Instances and Methodology

We tested our algorithm on four classes of problems. Three of them, TSP, ORLIB and ODM, have been previously studied in the literature for the p-median problem. The fourth, RW, is introduced here as a set of instances that benefit less from our methods.

Class TSP contains three sets of points on the plane (with cardinality 1400, 3038, and 5934), originally used in the context of the traveling salesman problem (Reinelt, 1991). In the p-median problem, each point is both a user to be served and a potential facility, and distances are Euclidean. Following Hansen et al. (2001), we tested several values of p for each instance, ranging from 10 to approximately $n/3$, when comparing our algorithm to Whitaker's.

Class ORLIB, originally introduced in Beasley (1985), contains 40 graphs with 100 to 900 nodes, each with a suggested value of p (ranging from 5 to 200). Each node

is both a user and a potential facility, and distances are given by shortest paths in the graph.

The instances in class ODM, proposed by Briant and Naddef (2004), model the *optimal diversity management problem*. In this problem, one must assemble a certain product that appears in a large number of configurations, each defined by the presence or absence of a certain number of features. Briant and Naddef give as an example the electrical wiring in cars. Assuming that setting up an assembly line for every possible configuration is not economically viable, only p configurations are actually produced. Requests for other configurations will be fulfilled by the least costly alternative that is compatible (i.e., contains all the necessary features) among those produced. The goal is to decide which p configurations to produce, given the demand and the unit cost for each existing configuration. To model this as a p-median problem, we make each configuration both a user and a facility. The cost of serving user u with facility f is the demand of u times the unit cost of f, as long as configuration f is compatible with configuration u; otherwise, the cost is infinity. We tested our algorithm on the four instances cited in Briant and Naddef (2004), with 535, 1284, 3773, and 5535 configurations. As in Briant and Naddef (2004), we tested values of p from 5 to 20 in each case.[6]

In class RW, each instance is a square matrix in which entry (u,f) is an integer taken uniformly at random from the interval $[1,n]$ and represents the cost of assigning user u to facility f. Four values of n were tested (100, 250, 500, and 1000), each with values of p ranging from 10 to $n/2$, totaling 27 combinations.[7] The random number generator we used when creating these instances (and in the algorithm itself) was Matsumoto and Nishimura's *Mersenne Twister* (Matsumoto and Nishimura, 1998).

Recall that the algorithms tested here use the distance oracle model, which assumes that retrieving the distance between any user and any facility takes $O(1)$ time. This can be trivially achieved for intances in RW (with a table look-up) and TSP (from the Euclidean coordinates). For ORLIB, we compute the all-pairs shortest paths in advance, as it is usually done in the literature (Hansen and Mladenović, 1997; Hansen et al., 2001). These computations are not included in the running times reported in this section, since they are the same for all methods (including Whitaker's). For ODM, to compute the distance between a user and a facility we need to know whether the user is covered by that facility or not. To answer this question in $O(1)$ time, we precompute an $n \times m$ boolean incidence matrix with this information. The same expected complexity could be achieved with a hash table, which potentially uses less space but has higher overhead for accessing each element. The time to build the incidence matrix is also not included in the times reported here.

All tests were performed on an SGI Challenge with 28 196-MHz MIPS R10000 processors (with each execution of the program limited to one processor) and 7.6 GB of memory. All algorithms were coded in C++ and compiled with the SGI MIPSpro

[6]In Briant and Naddef (2004), the authors do not show results for p greater than 16 in the instance with 3773 nodes. We include results for 17 to 20 as well, for symmetry.

[7]More precisely: for $n = 100$, we used $p = 10$, 20, 30, 40, and 50; for $n = 250$, $p = 10$, 25, 50, 75, 100, and 125; for $n = 500$, $p = 10$, 25, 50, 100, 150, 200, and 250; and for $n = 1000$, $p = 10$, 25, 50, 75, 100, 200, 300, 400, and 500.

C++ compiler (v. 7.30) with flags `-O3 -OPT:Olimit=6586`. The source code is available from the authors upon request, as are the RW instances.

All running times shown in this paper are CPU times, measured with the `getrusage` function, whose precision is 1/60 second. In some cases, actual running times were too small for this precision, so each algorithm was repeatedly run for at least 5 seconds. Overall times were measured, and averages reported here.

When comparing different local search methods, we applied them to the same initial solutions. These were obtained by two different algorithms. The first is greedy (Whitaker, 1983): starting from an empty solution, we insert one facility at a time, always picking the one that reduces the solution cost the most. The second algorithm is random: we just pick a set of p facilities uniformly at random as the initial solution. All tests with random solutions were repeated five times for each method, using five different random seeds.

Running times mentioned in this paper refer to the local search only, and they do not include the cost of building initial solution (which is the same for all methods).

10.6.2 Results

This section presents an experimental comparison of several variants of our implementation and Whitaker's method, *fast interchange* (we will use FI for short). We implemented FI based on the pseudocode in Hansen and Mladenović (1997) (obtaining comparable running times); the most important function was presented here in Figure 10.1.

10.6.2.1 Basic Algorithm (FM). We start with the most basic version of our implementation, in which *extra* is represented as a full (non-sparse) matrix. This version (called FM, for *full matrix*) already incorporates some acceleration, since calls to `updateStructures` are limited to affected users only. However, it does *not* include the accelerations suggested in Sections 10.4.3.2 (sparse matrix) and 10.4.3.3 (preprocessing).

To demonstrate that keeping track of affected users can lead to significant speedups, we devised the following experiment. We took one instance from each class: odm1284 (class ODM, 1284 nodes), pmed40 (class ORLIB, 900 nodes), fl1400 (class TSP, 1400 nodes), and rw1000 (class RW, 1000 nodes). Note that they all have a similar number of nodes. Each instance was tested with 99 different values of p, from 1% to 99% of m. Since for very large values of p the greedy algorithm almost always find local optima (thus rendering the local search useless), the initial solutions used in this experiment are random.

For each run, we computed how many calls to `updateStructures` and to `undoUpdateStructures` would have been made if we were not keeping track of affected users, and how many calls were actually made (in both cases, we did not count calls at the start of the first iteration, which is just the initialization). The ratio between these values, in percentage terms, is shown in Figure 10.4 (each point is the average of five runs).

It is clear that the average number of affected users is only a fraction of the total number of users, even for small values of p, and drops significantly as the number of facilities to open increases. In all four instances, the average number of affected users

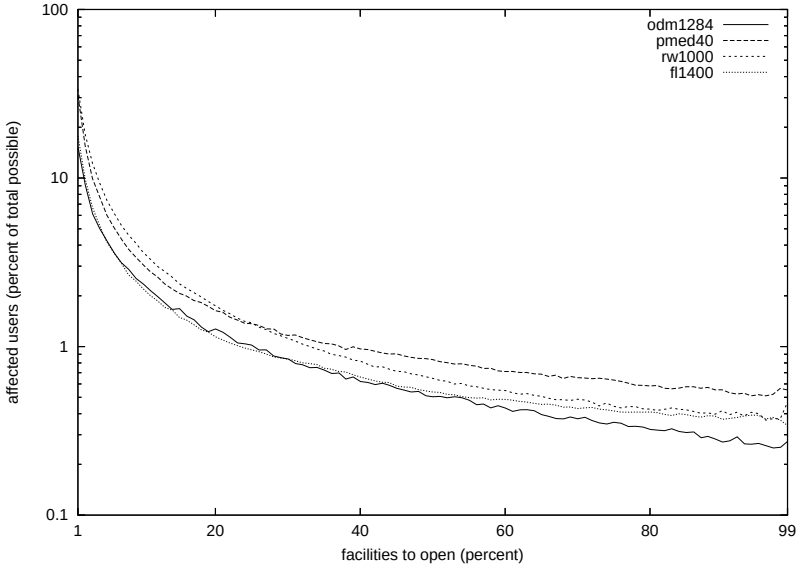


Figure 10.4 Percentage of users affected during a run of the local search as a function of p (the percentage is taken over the set of all possible users that could have been affected, considering all iterations). One instance in each class is represented. Vertical axis is in logarithmic scale.

eventually drops below 1% of n. By exploiting this fact, our implementation definitely has the potential to be faster than FI.

To test if this is indeed the case in practice, we ran an experiment with all instances from the four classes, with the values of p listed in Section 10.6.1. We used both greedy and random initial solutions. For each instance, we computed the speedup obtained by our method when compared to FI, i.e., the ratio between the running times of FI and FM. Table 10.1 shows the best, the (geometric) mean, and the worst speedups thus obtained considering all instances in each class.[8] Values greater than 1.0 favor our method, FM.

The table shows that even the basic acceleration scheme achieves speedups of up to more than 40. There are cases, however, in which FM is actually slower than Whitaker's method. This happens for instances in which the local search procedure performs very few iterations, insufficent to ammortize the overhead of using a matrix. This is more common with the greedy constructive heuristic, which is more likely to find solutions that are close to being local optima, particularly when p is very large or

[8]Since we are dealing with ratios, geometric (rather than arithmetic) means seem to be a more sensible choice; after all, if a method takes twice as much time for 50% of the instances and half as much for the other 50%, it should be considered roughly equivalent to the other method. Geometric means reflect that, whereas arithmetic means do not.

Table 10.1 Speedup obtained by FM (full matrix, no preprocessing) over Whitaker's FI.

SOLUTION	CLASS	BEST	MEAN	WORST
random	ODM	41.66	12.67	2.95
	ORLIB	21.19	5.76	1.64
	RW	20.96	7.62	2.51
	TSP	28.92	11.29	1.95
greedy	ODM	20.10	4.49	0.89
	ORLIB	14.20	3.76	1.07
	RW	13.99	5.50	1.47
	TSP	31.96	10.72	1.96

very small (the worst case among all instances happened with odm535 and $p = 6$). On average, however, FM has proven to be from three to more than ten times faster than FI.

10.6.2.2 Sparse Matrix (SM). We now analyze a second variant of our method. Instead of using a full matrix to represent *extra*, we use a sparse matrix, as described in Section 10.4.3.2. We call this variant SM. Recall that our rationale for using a sparse matrix was that the number of nonzero elements in the *extra* matrix is small. Figure 10.5 suggests that this is indeed true. For each of the four representative instances and each value of p (from 1% to 99% of m), it shows what fraction of the elements are nonzero (considering all iterations of the local search). The algorithm was run five times for each value of p, from five random solutions.

Although the percentage approaches 100% when the number of facilities to open is small, it drops very fast when p increases, approaching 0.1%. Note that rw1000, which is random, tends to have significantly more nonzeros for small values of p than other instances.

It is clear that the algorithm has a lot to benefit from representing only the nonzero elements of *extra*. However, the sparse matrix representation is much more involved than the array-based one, so some overhead is to be expected. Does it really reduce the running time of the algorithm in practice?

Table 10.2 shows that the answer to this question is "yes" most of the time. It represents the results obtained from all instances in the four classes, and contains the best, mean, and worst speedups obtained by SM over FI, for both types of initial solution (random and greedy).

As expected, SM has proven to be even faster than FM on average and in the best case (especially for the large instances with large values of p in the RW and TSP classes). However, some bad cases become slightly worse. This happens mostly for instances with small values of p: with a relatively large number of nonzero elements in the matrix, a sparse representation is not the best choice.

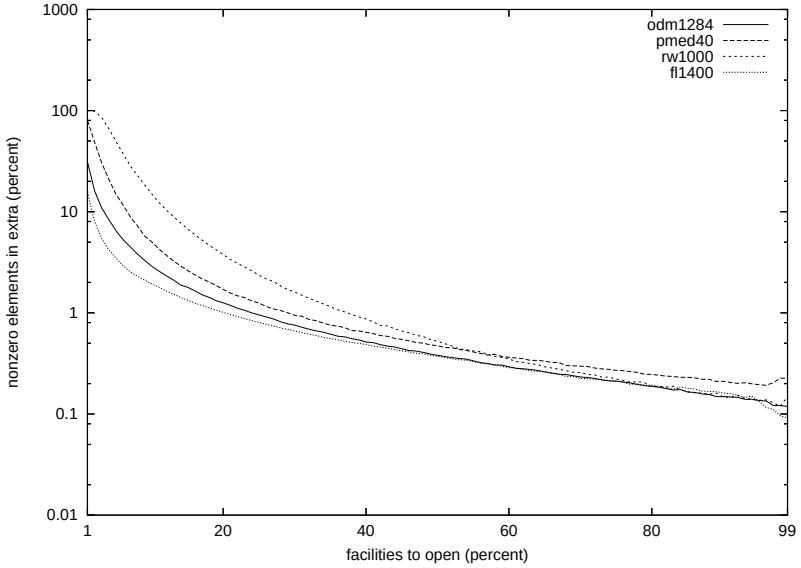


Figure 10.5 Percentage of entries in the extra matrix that have nonzero values as a function of p. One instance of each class is represented. Vertical axis is in logarithmic scale.

Table 10.2 Speedup obtained by SM (sparse matrix, no preprocessing) over Whitaker's FI.

SOLUTION	CLASS	BEST	MEAN	WORST
random	ODM	26.41	9.28	2.49
	ORLIB	46.88	6.66	1.19
	RW	114.36	12.47	1.95
	TSP	142.84	26.28	1.80
greedy	ODM	21.62	5.21	0.99
	ORLIB	24.88	4.36	1.00
	RW	49.35	8.36	1.22
	TSP	132.06	24.03	1.87

10.6.2.3 Sparse Matrix with Preprocessing (SMP). The last acceleration we study is the preprocessing step (Section 10.4.3.3), in which all potential facilities are sorted according to their distances from each of the users. We call this variant SMP, for *sparse matrix with preprocessing*. The goal of the acceleration is to avoid looping through all m facilities in each call to function `updateStructures` (and `undoUpdateStructures`). We just have to find the appropriate prefix of the ordered list.

Figure 10.6 shows the average size of the prefixes (as a percentage of m) that are actually checked by the algorithm, as a function of p (which varies from 1% to 99% of n). Initial solutions are random in this experiment.

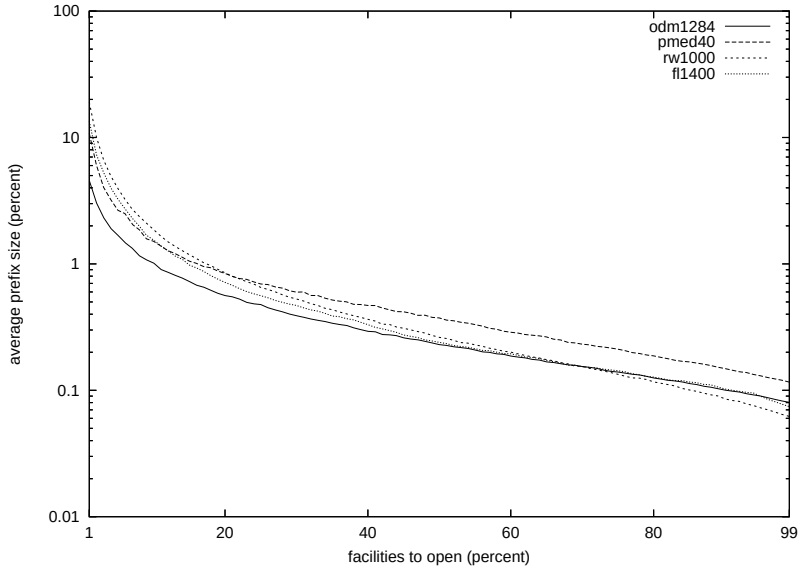


Figure 10.6 Percentage of facilities actually visited when updating structures, for several values of p. One instance of each class is represented. Vertical axis is in logarithmic scale.

As claimed before, the average prefix size is only tiny a fraction of m, for all but very small values of p. Considering only those prefixes instead of all facilities can potentially accelerate the local search. Of course, this does not come for free: the cost of preprocessing must be accounted for.

To determine the overall effect of these two conflicting factors, we tested SMP on all instances of our set. Table 10.3 shows the best, mean, and worst speedups obtained with respect to FI. Columns 3, 4, and 5 consider running times of the local search procedure only; columns 6, 7, and 8 also include preprocessing times.

The table shows that the entire SMP procedure (including preprocessing) is on average still much faster than Whitaker's FI, but often slightly slower than the other variants studied in this paper (FM and SM). However, as already mentioned, metaheuristics often need to run the local search procedure several times, starting from different solutions. Since preprocessing is run only once, its cost can be quickly amortized. Columns 3, 4, and 5 of the table show that once this happens, SMP can achieve truly remarkable speedups with respect not only to FI, but also to other variants studied in this paper. In the best case (instance rl5934 with $p = 800$), it is roughly 800 times faster than FI.

To evaluate how fast the amortization is, consider what would happen in a simple multistart procedure. In each iteration, this algorithm generates a random solution and

Table 10.3 Speedup obtained by SMP (sparse matrix, full preprocessing) over Whitaker's FI.

SOLUTION	CLASS	LOCAL SEARCH ONLY			INCLUDING PREPROCESSING		
		BEST	MEAN	WORST	BEST	MEAN	WORST
random	ODM	46.18	13.77	3.42	8.26	3.00	0.87
	ORLIB	77.44	8.75	1.28	22.42	3.40	0.66
	RW	169.59	17.51	1.92	48.37	6.26	1.05
	TSP	812.80	186.81	4.63	128.03	31.92	1.89
greedy	ODM	33.16	7.21	1.33	3.30	0.67	0.15
	ORLIB	43.26	6.40	1.37	6.86	1.10	0.21
	RW	91.05	12.59	1.34	9.98	2.14	0.20
	TSP	695.57	161.86	5.11	71.42	18.92	1.45

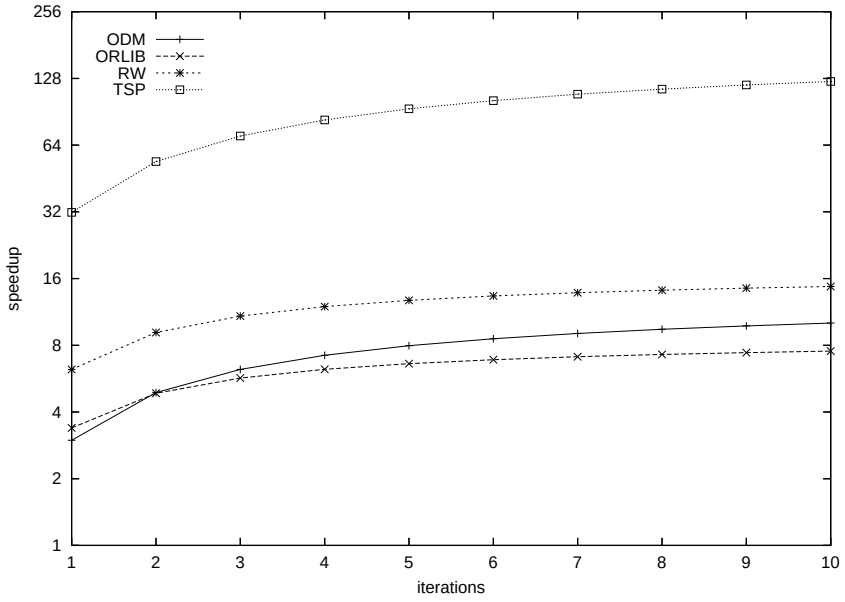


Figure 10.7 Speedup of a multistart procedure implemented with SMP with respect to those an implementation using Whitaker's method (FI).

applies local search to it; the best solution found over all iterations is picked. We can predict the behavior of such a method (as far as running times are concerned) from the data used to build Table 10.3. After only one iteration, the mean speedups obtained when SMP is used instead of FI (Whitaker's method) will be those shown in the seventh column of the table. As the number of iterations increases, the mean

speedups will gradually converge to the values in the fourth column. Figure 10.7 shows exactly what happens as a function of the number of iterations. After only ten iterations, the speedups are already close to those shown in the fourth column of Table 10.3: 10.1 for ODM, 7.5 for ORLIB, 14.7 for RW, and 124.0 for TSP.

Apart from the preprocessing time, another important downside of strategy SMP is memory usage: an array of size m is kept for each of the n customers. As mentioned in Section 10.4.3.3, one can use less memory by storing a vector with only a fraction of the m facilities for each customer. Table 10.4 shows what happens when we restrict the number of elements per vector to $5m/p$; we call this version of the local search SM5. In general, SMq is an algorithm that associates a list with qm/p facilities with each user. We use m/p as a parameter because this correlates well with the number of facilities each user has to look at to find an open one.

Table 10.4 Speedup obtained by SM5 (sparse matrix, with preprocessing, cache size $5p/m$) over Whitaker's FI.

SOLUTION	CLASS	LOCAL SEARCH ONLY			INCLUDING PREPROCESSING		
		BEST	MEAN	WORST	BEST	MEAN	WORST
random	ODM	46.12	13.68	3.42	14.48	4.04	0.86
	ORLIB	77.42	8.81	1.29	40.14	4.52	0.66
	RW	166.51	17.44	2.01	93.08	9.57	1.13
	TSP	774.96	176.42	4.49	283.71	62.97	2.20
greedy	ODM	32.65	7.16	1.30	6.23	0.96	0.14
	ORLIB	44.31	6.41	1.33	14.51	1.61	0.20
	RW	92.93	12.62	1.34	24.73	3.87	0.22
	TSP	747.72	160.93	5.07	177.62	40.65	1.73

Tables 10.3 and 10.4 show that using restricted lists (as opposed to m-sized ones) can make the algorithm significantly faster when preprocessing times are considered. This is true especially for large instances. On average, SM5 is roughly twice as fast as SMP. The gains from a faster preprocessing more than offset the potential extra time incurred during the actual local search. In fact, the table also shows that the time spent on the main loop is barely distinguishable from SMP; the partial lists are almost always enough for the algorithm. Local search within SM5 can actually be slightly *faster* than within SMP. The possible cause here are cache effects; since less data is kept in memory, there is more locality to be exploited by the hardware.

10.6.2.4 Overall Comparison. To get a better understanding of the performance of all variants proposed in this paper, we study in detail the largest instance in our set (rl5934, with almost 6000 customers and facilities). Figures 10.8 and 10.9 show the running times of several methods (FI, FM, SM, SM1, SM2, SM3, SM5, and SMP) for different values of p. Times are averages of five runs from different random solutions (the same set of initial solutions was given to each method). The first figure

considers the local search only, whereas the second accounts for preprocessing times as well.

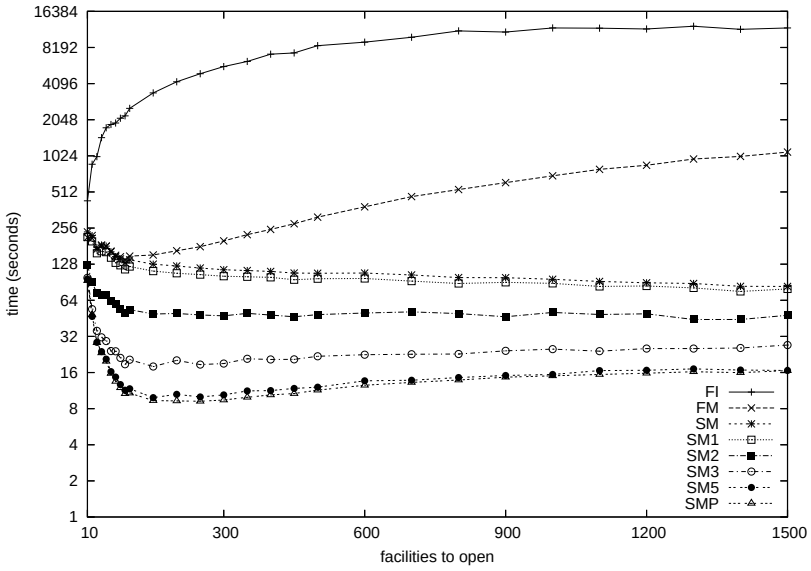


Figure 10.8 Instance rl5934: dependency of running times on p for different methods. Times are in logarithmic scale and do not include preprocessing.

The figures show that for some methods, such as Whitaker's FI and the full-matrix variant of our implementation (FM), an increase in p leads to greater running times (although our method is still 10 times faster for $p = 1500$). For all other methods, which use sparse matrices, the time spent per iteration tends to decrease as p increases: the effect of swaps becomes more local, with fewer users affected and fewer neighboring facilities visited in each call to `updateStructures`. This latter effect explains why keeping even a relatively small list of neighboring facilities for each user seems to be worthwhile. The curves for variants SMP and SM5 are practically indistinguishable in Figure 10.8, and both are much faster than SM (which keeps no list at all).

As a final note, we observe that, because all methods discussed here implement the same algorithm, the number of iterations does not depend on the method itself. It does, however, depend on the value of p: in general, these two have a positive correlation for $p \leq m/2$, and negative from this point on, as Figure 10.10 shows. This correlates well with the total number of solutions: there are $\binom{m}{p}$ solutions of size p, and this expression is maximized for $p = m/2$.

10.6.2.5 Profile. The results for SMP show that the modifications proposed in this paper can, together, result in significant acceleration. How much further can we go? Can additional modifications to the algorithm make it even faster?

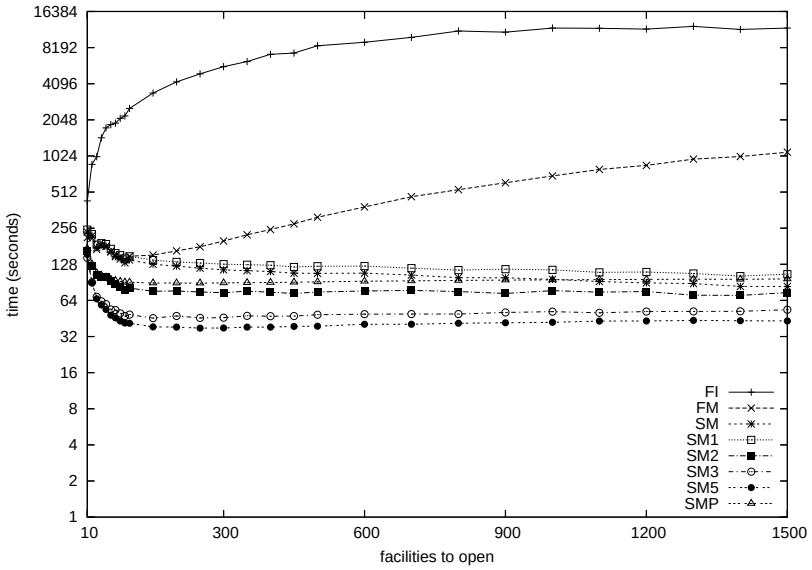


Figure 10.9 Instance rl5934: dependency of running times on p for different methods. Times are in logarithmic scale and include preprocessing where applicable.

Table 10.5 Execution profile for method SMP: percentage of time spent on each of the potential bottlenecks (only the largest instance in each class is shown). Preprocessing times are not considered.

INSTANCE			INIT.	UPDATE	UPDATE	BEST	OTHER
NAME	n, m	p		CLOSEST	STRUCT.	NEIGH.	OPER.
odm5535	5535	56	17.7	5.9	62.3	7.8	6.2
		1384	6.4	19.7	4.5	30.9	38.5
pmed40	900	9	6.7	1.7	89.8	0.6	1.2
		225	13.4	29.4	13.5	11.2	32.5
rw1000	1000	10	3.7	1.4	93.7	0.5	0.7
		250	12.1	26.7	15.1	14.5	31.6
rl5934	5934	60	12.2	5.7	74.0	5.0	3.1
		1484	10.7	41.0	4.6	22.7	21.0

These are open questions. However, we argue that small modifications are unlikely to lead to major gains, particularly when p is large. To support this claim, we devised the following experiment. For each class, we took the instance with the greatest number of users (n) and ran SMP with two values of p ($0.01n$ and $0.25n$), from five random solutions in each case. Table 10.5 shows the percentage of the total local

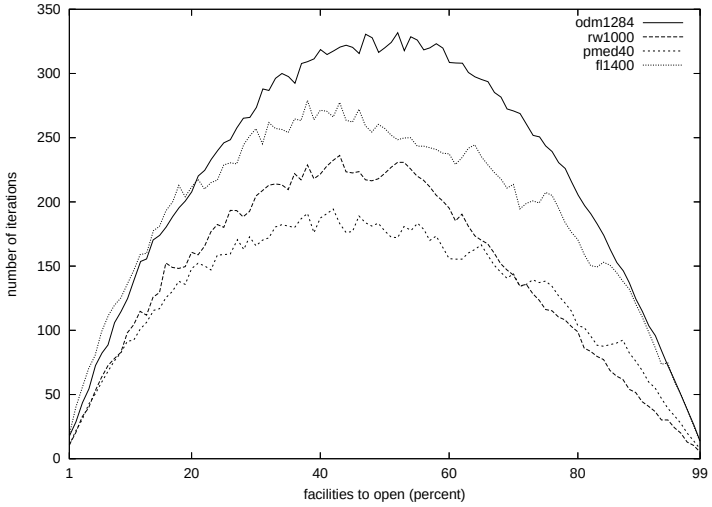


Figure 10.10 Number of iterations of the local search procedure as a function of p, starting from random solutions. One instance from each class is represented.

search time (excluding preprocessing) spent in each section of the algorithm: initialization (which includes allocating the data structures), calls to `updateClosest`, calls to `updateStructures` (and `undoUpdateStructures`), calls to `bestNeighbor`, and other operations (such as determining which users are affected).

Note that calls to `updateStructures` and `undoUpdateStructures` dominate the running time for small values of p. This is to be expected: these functions run in $O(mn)$ time, while `bestNeighbor` and `updateClosest` run in $O(pn)$ and $O(pm)$ operations, respectively. When p increases, the running time for `updateStructures` and `undoUpdateStructures` actually decreases, since a larger fraction of the elements in the *extra* matrix will be zero (and therefore will not need to accessed). As a result, no component took more than 50% of the running time for $p = 0.25n$. In this case, even if we could make a component run in virtually no time, the algorithm would be at most twice as fast. A decent speedup, but not at all comparable to 800, the factor we were able to achieve in this paper. To obtain better factors, it seems necessary to work on all bottlenecks at once, or to come up with a different strategy altogether.

10.7 CONCLUDING REMARKS

We have presented a new implementation of the swap-based local search for the p-median problem introduced by Teitz and Bart. We combine several techniques (using a matrix to store partial results, a compressed representation for this matrix, and preprocessing) to obtain speedups of up to three orders of magnitude with respect to the best previously known implementation, due to Whitaker. Our implementation is especially well suited to relatively large instances with moderate to large values of p

and, due to the preprocessing step, to situations in which the local search procedure is run several times for the same instance (such as within a metaheuristic). When the local search has very few iterations, Whitaker's method can still be faster if the preprocessing time is considered.

An important test to the algorithms proposed here would be to apply them within more sophisticated metaheuristics. We have done that in Resende and Werneck (2004). That paper describes a multistart heuristic for the p-median problem that relies heavily on local search and path-relinking, both implemented according to the guidelines detailed in this paper. The algorithm has proved to be very effective in practice, obtaining remarkably good results (in terms of running times and solution quality) when compared to other methods in the literature.

A possible extension of our work presented would be to apply the methods and ideas presented here to problems beyond p-median and facility location. Swap-based local search is a natural procedure to be performed on problems such as maximum set cover, for example.

Acknowledgements.. We thank two anonymous referees for their helpful comments. We also thank Dennis Naddef for providing us with the instances in the ODM class.

Bibliography

V. Arya, N. Garg, R. Khandekar, A. Mayerson, K. Munagala, and V. Pandit. Local search heuristics for *k*-median and facility location problems. In *Proc. 33rd ACM Symposium on the Theory of Computing*, 2001.

P. Avella, A. Sassano, and I. Vasil'ev. Computational study of large-scale *p*-median problems. Technical Report 08-03, DIS — Università di Roma "La Sapienza", 2003a.

P. Avella, A. Sassano, and I. Vasil'ev. A heuristic for large-scale *p*-median instances. *Electronic Notes in Discrete Mathematics*, 13:1–4, 2003b.

J. E. Beasley. A note on solving large *p*-median problems. *European Journal of Operational Research*, 21:270–273, 1985.

O. Briant and D. Naddef. The optimal diversity management problem. *Operations Research*, 52(4):515–526, 2004.

T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

G. Cornuéjols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: An analytical study of exact and approximate algorithms. *Management Science*, 23:789–810, 1977.

O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194:229–237, 1999.

R. D. Galvão. A dual-bounded algorithm for the *p*-median problem. *Operations Research*, 28:1112–1121, 1980.

F. García-López, B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega. Parallelization of the scatter search for the *p*-median problem. *Parallel Computing*, 29(5):575–589, 2003.

F. Glover. Tabu search and adaptive memory programming: Advances, applications and challenges. In R. S. Barr, R. V. Helgason, and J. L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

M. F. Goodchild and V. Noronha. Location-allocation for small computers. Monograph 8, Department of Geography, University of Iowa, 1983.

P. Hansen and N. Mladenović. Variable neighborhood search for the *p*-median. *Location Science*, 5:207–226, 1997.

P. Hansen, N. Mladenović, and D. Perez-Brito. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(3):335–350, 2001.

M. J. Hodgson. Toward more realistic allocation in location-allocation models: An interaction approach. *Environment and Planning A*, 10:1273–85, 1978.

O. Kariv and L. Hakimi. An algorithmic approach to nework location problems, Part II: The *p*-medians. *SIAM Journal of Applied Mathematics*, 37(3):539–560, 1979.

A. A. Kuehn and M. J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9(4):643–666, 1963.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

F. E. Maranzana. On the location of supply points to minimize transportation costs. *Operations Research Quarterly*, 15(3):261–270, 1964.

M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

G. Reinelt. TSPLIB: A traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991. *http://www.iwr.uni-heidelberg.de/groups/comopt/software/-TSPLIB95/.*

M. G. C. Resende and C. C. Ribeiro. GRASP with path-relinking: Recent advances and applications. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*. Kluwer, 2005. In press.

M. G. C. Resende and R. F. Werneck. On the implementation of a swap-based local search procedure for the *p*-median problem. In R. E. Ladner, editor, *Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments (ALENEX'03)*, pages 119–127. SIAM, 2003.

M. G. C. Resende and R. F. Werneck. A hybrid heuristic for the *p*-median problem. *Journal of Heuristics*, 10(1):59–88, 2004.

E. Rolland, D. A. Schilling, and J. R. Current. An efficient tabu search procedure for the *p*-median problem. *European Journal of Operational Research*, 96:329–342, 1996.

K. E. Rosing. An empirical investigation of the effectiveness of a vertex substitution heuristic. *Environment and Planning B*, 24:59–67, 1997.

K. E. Rosing and C. S. ReVelle. Heuristic concentration: Two stage solution construction. *European Journal of Operational Research*, 97:75–86, 1997.

K. E. Rosing, C. S. ReVelle, and H. Rosing-Vogelaar. The *p*-median and its linear programming relaxation: An approach to large problems. *Journal of the Operational Research Society*, 30(9):815–823, 1979.

E. L. F. Senne and L. A. N. Lorena. Langrangean/surrogate heuristics for *p*-median problems. In M. Laguna and J. L. González-Velarde, editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pages 115–130. Kluwer, 2000.

E. L. F. Senne and L. A. N. Lorena. Stabilizing column generation using Lagrangean/surrogate relaxation: an application to *p*-median location problems. *European Journal of Operational Research*, 2002. To appear.

E. L. F. Senne, L. A. N. Lorena, and Marcos A. Pereira. A branch-and-price approach to *p*-median location problems. *Computers and Operations Research*, 32:1655–1664, 2005.

E. D. Taillard. Heuristic methods for large centroid clustering problems. *Journal of Heuristics*, 9(1):51–74, 2003.

M. B. Teitz and P. Bart. Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, 16(5):955–961, 1968.

M. Thorup. Quick *k*-median, *k*-center, and facility location for sparse graphs. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP 2001)*, volume 2076 of *Lecture Notes in Computer Science*, pages 249–260. Springer, 2001.

S. Voß. A reverse elimination approach for the *p*-median problem. *Studies in Locational Analysis*, 8:49–58, 1996.

R. Whitaker. A fast algorithm for the greedy interchange of large-scale clustering and median location prolems. *INFOR*, 21:95–108, 1983.

11 A HYBRID HEURISTIC FOR THE P-MEDIAN PROBLEM

Mauricio G. C. Resende[1] and Renato F. Werneck[2]

[1]Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA.
mgcr@research.att.com

[2]Department of Computer Science
Princeton University
Princeton, NJ 08544 USA
rwerneck@cs.princeton.edu

Abstract: Given n customers and a set F of m potential facilities, the p-median problem consists in finding a subset of F with p facilities such that the cost of serving all customers is minimized. This is a well-known NP-complete problem with important applications in location science and classification (clustering). We present a multistart hybrid heuristic that combines elements of several traditional metaheuristics to find near-optimal solutions to this problem. Empirical results on instances from the literature attest the robustness of the algorithm, which performs at least as well as other methods, and often better in terms of both running time and solution quality. In all cases the solutions obtained by our method were within 0.1% of the best known upper bounds.
Keywords: p-median, hybrid heuristic, GRASP, path-relinking, metaheuristics, local search.

11.1 INTRODUCTION

The p-median problem is defined as follows. Given a set F of m potential facilities, a set U of n users (or customers), a distance function $d : U \times F \rightarrow \mathcal{R}$, and a constant $p \leq m$, determine which p facilities to open so as to minimize the sum of the distances from each user to its closest open facility. It is a well-known NP-hard problem (Kariv and Hakimi, 1979), with numerous applications in location science (Tansel et al., 1983) and classification (clustering) (Rao, 1971; Vinod, 1969).

Several algorithms for the *p*-median problem have been proposed, including exact methods based on linear programming (Beasley, 1985; Cornuejols et al., 1977; Galvão, 1980; Rosing et al., 1979), constructive algorithms (Cornuejols et al., 1977; Kuehn and Hamburger, 1963; Taillard, 2003; Whitaker, 1983), dual-based algorithms (Galvão, 1980; Nauss and Markland, 1981), and local search procedures (Goodchild and Noronha, 1983; Hodgson, 1978; Maranzana, 1964; Resende and Werneck, 2003; Rosing, 1997; Taillard, 2003; Teitz and Bart, 1968; Whitaker, 1983). Recently, metaheuristics capable of obtaining solutions of near-optimal quality have been devised. Tabu search procedures have been proposed by Voß (1996) and Rolland et al. (1996). The latter method was compared by Rosing et al. (1998) with the *heuristic concentration* method (Rosing and ReVelle, 1997), which obtained comparatively superior results. Hansen and Mladenović (1997) proposed a VNS (variable neighborhood search) for the problem, later parallelized by García-López et al. (2002). A variation of this method, VNDS (variable neighborhood decomposition search), was suggested by Hansen et al. (2001). Heuristics based on linear programming were studied by du Merle et al. (1999) and by Senne and Lorena (2000) and Senne and Lorena (2002).

In this chapter, we propose a hybrid heuristic for the *p*-median problem. In essence, it is a multistart iterative method, each iteration of which consists of the randomized construction of a solution, which is then submitted to local search. Traditionally, a multistart algorithm takes the best solution obtained in all iterations as its final result. Our method enhances this basic approach with some intensification strategies. We keep a pool with some of the best solutions found in previous iterations, the so-called *elite solutions*. In each iteration of our procedure, the solution obtained by local search is combined with one elite solution through a process called *path-relinking* (Glover, 1996; Glover et al., 2000; Laguna and Martí, 1999). Furthermore, after all iterations of the multistart phase are completed, we have a second, post-optimization phase in which elite solutions are combined with each other. Figure 12.1 summarizes our method, to which we will refer as HYBRID.

Note that this algorithm combines elements of several "pure" metaheuristics. Like GRASP (greedy randomized adaptive search procedure), our method is a multistart approach in which each iteration consists basically of a randomized greedy procedure followed by local search (Feo and Resende, 1989; 1995; Resende and Ribeiro, 2003). From tabu search and scatter search, our method borrows the idea of path-relinking (Glover, 1996; Laguna and Martí, 1999). Moreover, as Section 11.5 shows, we augment path-relinking with the concept multiple generations, a key feature of genetic algorithms (Goldberg, 1989; Michalewicz, 1994).

Of course, much remains to be specified to turn the outline in Figure 12.1 into an actual algorithm. We study each individual component (constructive algorithm, local search, and intensification) separately in Sections 11.3, 11.4, and 11.5, respectively. In Section 11.6, we present the results obtained by the final version of our method and compare them with those obtained by other methods in the literature. But first, in Section 11.2, we discuss important aspects of the experiments we conducted to evaluate individual components and to produce the final results.

```
function HYBRID (seed, maxit, elitesize)
1     randomize(seed);
2     init(elite, elitesize)
3     for i = 1 to maxit do
4           S ← randomizedBuild();
5           S ← localSearch(S);
6           S' ← select(elite, S);
7           if (S' ≠ NULL) then
8                 S' ← pathRelinking(S, S');
9                 add(elite, S');
10          endif
11          add(elite, S);
12    endfor
13    S ← postOptimize(elite);
14    return S;
end HYBRID
```

Figure 11.1 Pseudocode for HYBRID.

11.2 TESTING

11.2.1 Instances

We tested our algorithm on five classes of problems: TSP, ORLIB, SL, GR, and RW.

Instances in class TSP are just sets of points on the plane. Originally proposed for the traveling salesman problem, they are available from the TSPLIB (Reinelt, 1991). In the context of the p-median problem, they were first used by Hansen and Mladenović (1997); Hansen et al. (2001). Every point is considered both a potential facility and a customer, and the cost of assigning customer c to facility f is simply the Euclidean distance between the points representing c and f. Following Hansen et al. (2001), we consider three instances (fl1400, pcb3038, and rl5934, with 1400, 3038, and 5934 nodes, respectively), each with several different values for p (number of facilities to open).

Class ORLIB (short for OR-Library) was introduced by Beasley (1985). Each of the 40 instances (pmed01, pmed02, ..., pmed40) in the class is a graph with a corresponding value for p. Every node is a customer and a potential facility, and the cost of assigning a customer to a facility is the length of the shortest path between the corresponding nodes. The number of nodes in this class varies from 100 to 900, and the value of p from 5 to 200.

The third class we consider is SL, a slight extension to ORLIB proposed by Senne and Lorena (2000). It contains three new instances, all based on graphs from ORLIB: sl700 uses the same graph as pmed34, but with $p = 233$; sl800 is the same as pmed37, with $p = 267$; and sl900 is pmed40 with $p = 300$ (Senne, 2002).

The fourth class studied is GR, introduced by Galvão and ReVelle (1996) and first used for the p-median problem by Senne and Lorena (2000). This class contains two graphs, with 100 and 150 nodes (named gr100 and gr150, respectively). Eight values of p (between 5 and 50) were considered in each case.

The fifth class we study is RW. Originally proposed by Resende and Werneck (2003), it corresponds to completely random distance matrices. In every case, the number of potential facilities (m) is equal to the number of customers (n). The distance between each facility and each customer has an integer value taken uniformly at random from the interval $[1,n]$.[1] Four different values of n were considered: 100, 250, 500, and 1000 (instance names are rw100, rw250, rw500, and rw1000, respectively). In each case, several values of p were tested.

Costs are integral in all classes except TSP, in which distances are, in theory, real values. We did not explicitly round nor truncate values, which were kept with `double` precision throughout the algorithm.

Results obtained by the final version of our algorithm on all instances are shown in Section 11.6. We also conducted experiments with several variants of our method to assess how each individual component (constructive algorithm, local search, and path-relinking, among others) affects the overall performance. In those experiments, however, we used only a *restricted set* of instances. This set was chosen with two goals in mind. First, its instances should be hard enough to reveal the differences between various parameters and components. Some instances, especially in class ORLIB, can be solved to optimality by local search alone, thus making it pointless to include them in comparative tests. Our second goal was to keep the set small enough so as to allow statistically meaningful experiments (i.e., with several pseudorandom number generator seeds for each instance) on a relatively small amount of CPU time (no more than a few days per experiment). Given those goals and our experience from early versions of the algorithm, we defined the restricted set with 10 instances: pmed15 and pmed40, both from class ORLIB; sl700, from class SL; fl1400 (with $p = 150$ and $p = 500$) and pcb3038 (with $p = 30$ and $p = 250$), from class TSP; gr150 (with $p = 25$), from class GR; and rw500 (with $p = 25$ and $p = 75$), from class RW.

11.2.2 Testing Environment

Tests were performed on an SGI Challenge with 28 196-MHz MIPS R10000 processors (with each execution of the program limited to only one processor), 7.6 GB of memory, and IRIX 5 as the operating system. The algorithm was coded in C++ and compiled with the SGI MIPSpro C++ compiler (v. 7.30) with flags `-O3 -OPT:Olimit=6586`. All running times shown in this chapter are CPU times, measured with the `getrusage` function. Running times do not include the time spent reading instances from disk, but they do include the computation of all-pairs shortest paths on graph instances (classes SL and ORLIB).[2] The pseudorandom number generator we use is *Mersenne Twister* (Matsumoto and Nishimura, 1998).

[1] In particular, unlike all other classes, the distance from facility i to user i is not zero. Moreover, the distance between facility i and user j need not be equal to the distance between facility j and user i.

[2] GR is also a graph-based class, but the instances we obtained, kindly provided by E. Senne, were already represented as distance matrices.

11.3 CONSTRUCTIVE ALGORITHMS

The standard greedy algorithm for the p-median problem (Cornuejols et al., 1977; Whitaker, 1983) starts with an empty solution and adds facilities one at a time, choosing the most profitable in each iteration (the one whose insertion causes the greatest drop in solution cost). Evidently, this method cannot be used directly in our algorithm: being completely deterministic, it would yield identical solutions in all iterations. We considered the following randomized variants in our experiments:

- random (random solution): Select p facilities uniformly at random. The selection itself requires $O(m)$ time, and determining which facility should serve each customer requires $O(pn)$ operations.[3] Therefore, the overall complexity of the algorithm is $O(m+pn)$.

- rpg (random plus greedy): Select a fraction α (an input parameter) of the p facilities at random, then complete the solution in a greedy fashion. The algorithm takes $O((m+\alpha pn)+(1-\alpha)(pmn))$ time in the worst case, which corresponds to $O(pmn)$ if α is not very close to 1. In our tests, a value for α was chosen uniformly at random in the interval $[0;1]$ in every multistart iteration.

- rgreedy (randomized greedy): Similar to the greedy algorithm, but in each step i, instead of selecting the best among all $m-i+1$ options, choose randomly from the $\lceil \alpha(m-i+1) \rceil$ best options, where $0 < \alpha \leq 1$ is an input parameter. Note that if $\alpha \to 0$, this method degenerates into the greedy algorithm; if $\alpha \to 1$, it turns into the random algorithm. In our tests, we selected α uniformly at random in the interval $(0;1]$ in each iteration of the multistart phase. This algorithm takes $O(pmn)$ time.

- pgreedy (proportional greedy): Yet another variant of the greedy algorithm. In each iteration, compute, for every candidate facility f_i, how much would be saved if f_i were added to the solution. Let $s(f_i)$ be this value. Then pick a candidate at random, but in a biased way: the probability of a given facility f_i being selected is proportional to $s(f_i) - \min_j s(f_j)$. If all candidates are equally good (they would all save the same amount), select one uniformly at random. This method also takes $O(pmn)$ time.

- pworst (proportional worst): In this method, the first facility is selected uniformly at random. Other facilities are added one at a time as follows. Compute, for each customer, the difference between how much its current assignment costs and how much the best assignment would cost; then select a customer at random, with probability proportional to this value, and open the closest facility. Customers for which the current solution is particularly bad have a greater chance of being selected. This method, also used by Taillard (2003), runs in $O(mn)$ time in the worst case.

[3] This can be made faster in some settings, like sparse graphs or points on the Euclidean plane. The results in this chapter, however, do not use any such metric-specific accelerations.

- sample (sample greedy): This method is similar to the greedy algorithm, but instead of selecting the best among all possible options, it only considers $q < m$ possible insertions (chosen uniformly at random) in each iteration. The most profitable among those is selected. The running time of the algorithm is $O(m + pqn)$. The idea is to make q small enough so as to significantly reduce the running time of the algorithm (when compared to the pure greedy one) and to ensure a fair degree of randomization. In our tests, we used $q = \lceil \log_2(m/p) \rceil$.

We note that a "pure" multistart heuristic would use random as the constructive algorithm. Method rgreedy, which selects a random element from a restricted list of candidates, would be the one used by a standard GRASP. All other methods are meant to be faster variants of rgreedy.

It was not clear at first which of these methods would be most adequate as a building block of our heuristic. To better analyze this issue, we conducted an experiment on the restricted set of instances defined in Section 11.2.1. For every instance in the set and every constructive procedure, we ran our heuristic 10 times, with 10 different seeds. In every case, the number of iterations was set to 32, with 10 elite solutions, using up:down as the criterion to determine the direction of path-relinking (this criterion is defined in Section 11.5.4.2).

To explain the results shown in Table 11.1, we need some additional definitions. For each instance, we compute the overall average solution value obtained by all 60 executions of HYBRID (6 different methods, each with 10 seeds). Then, for each method, we determine the *relative percentage deviation* for that instance: how much the average solution value obtained by that method is above (or below) the overall average in percentage terms. By taking the average of these deviations over all 10 instances, we obtain the *average relative percentage deviation* (%DEV) for each method; these are the values shown in column 2 of Table 11.1. Column 4 was computed in a similar fashion, but considering running times instead of solution values.

Columns 3 and 5 were computed as follows. For each instance, the methods were sorted according to their relative percentage deviations; the best received one point, the second two points, and so on, until the sixth best method, with six points. When there was a tie, points were divided equally between the methods involved. For example, if the deviations were -0.03, -0.02, -0.02, 0.01, 0.03, and 0.03, the corresponding methods would receive $1, 2.5, 2.5, 4, 5.5,$ and 5.5 points, respectively. The number of points received by a method according to this process is its *rank* for that particular instance. Its *normalized rank* was obtained by linearly mapping the range of ranks (1 to 6, in this case) to the interval $[-1, 1]$. In the example above, the normalized ranks would be $-1, -0.4, -0.4, 0.2, 0.8,$ and 0.8. The normalized ranks must add up to zero (by definition). If a method is always better than all others, its normalized rank will be -1; if always worse, it will be 1. What columns 3 and 5 of Table 11.1 show are the *average normalized ranks* of each method, taken over the set of 10 instances. Column 3 refers to solution quality, and column 5 to running times.

The correlation between these measures is higher when one method is obviously better (or worse) than other methods. In general, however, having a lower average normalized rank does not imply having a better average relative percentage deviation, as the table shows.

Table 11.1 HYBRID results with different constructive procedures: Average relative percentage deviations (%DEV) and average normalized ranks (NRANK) for solution qualities and running times (both referring to the entire HYBRID procedure). Smaller values are better.

METHOD	QUALITY		TIME	
	%DEV	NRANK	%DEV	NRANK
pgreedy	-0.009	0.160	39.6	0.920
pworst	-0.006	-0.400	-18.7	-0.480
rgreedy	0.020	-0.160	35.8	0.400
random	0.015	0.000	-24.9	-0.840
rpg	0.009	0.620	-12.3	-0.300
sample	-0.029	-0.220	-19.5	-0.600

It is clear that the methods are distinguishable much more by running time than by solution quality. As the analysis of their worst case complexities suggests, rgreedy and pgreedy are much slower than the other methods. In fact, they are so much slower that, as shown in the table, they make the entire HYBRID heuristic take twice as long on average than when using faster methods. The other method with $O(pmn)$ worst-case performance, rpg, while much faster than rgreedy and pgreedy in practice, is still slower than other methods without finding better solutions on average. We therefore tend to favor the three relatively fast methods: pworst, sample, and random. Among those, sample and pworst seem to lead to solutions of slightly better quality. We chose sample for the final version of our algorithm, although pworst would probably find very similar results.

This experiment reveals an unusual feature of the p-median problem. In the GRASP framework (upon which our heuristic is based), the running time of the randomized greedy algorithm is usually not an issue. The randomized constructive methods should produce solutions that are as good as possible given the diversity constraints, thus reducing the number of iterations of the generally much slower local search. In our case, the local search is relatively so fast that investing extra time in building the solution can actually make the whole algorithm much slower without any significant gain in terms of solution quality. We could not apply the randomization strategy normally used in GRASP, represented here by rgreedy. Instead, we had to develop a faster alternative based on sampling. That is why we call our method a *hybrid heuristic* instead of GRASP.[4]

11.4 LOCAL SEARCH

The standard local search procedure for the p-median problem, originally proposed by Teitz and Bart (1968) and studied or used by several authors (García-López et al., 2002; Hansen and Mladenović, 1997; Hansen et al., 2001; Hodgson, 1978; Resende and Werneck, 2003; Whitaker, 1983), is based on swapping facilities. Given an initial

[4]An earlier version of this chapter (Resende and Werneck, 2002) did refer to the algorithm as "GRASP with path-relinking". We believe that "hybrid heuristic" is a more accurate characterization.

solution S, the procedure determines, for each facility $f \notin S$, which facility $g \in S$ (if any) would improve the solution the most if f and g were interchanged (i.e., if f were opened and g closed). If there is one such improving move, f and g are interchanged. The procedure continues until no improving interchange can be made, in which case a *local minimum* will have been found.

Whitaker (1983) proposed an efficient implementation of this method, which he called *fast interchange*. A similar implementation was used by Hansen and Mladenović (1997) and, later, in other papers (García-López et al., 2002; Hansen et al., 2001). A minor difference between them is the fact that Whitaker adopts a *first improvement* strategy (the algorithm moves to a neighboring solution as soon as it finds an improving one), while the others prefer *best improvement* (all neighbors are checked and the very best is chosen). In either case, the running time of each iteration is bounded by $O(mn)$.

Resende and Werneck (2003) have recently proposed an alternative implementation, also using best improvement. Although it has the same worst-case complexity as Whitaker's, it can be substantially faster in practice. The speedup (of up to three orders of magnitude) results from the use of information gathered in early iterations of the algorithm to reduce the amount of computation performed in later stages. This, however, requires a greater amount of memory. While Whitaker's implementation requires $O(n)$ memory in the worst case (not considering the distance matrix), the alternative may use up to $O(mn)$ memory positions.

In any case, we believe that the speedup is well worth the extra memory requirement. This is especially true for methods that rely heavily on local search procedures. This includes not only multistart methods such as the one described here, but also VNS (Hansen and Mladenović, 1997) and tabu search (Rolland et al., 1996; Voß, 1996), for example. Furthermore, one should also remember that while the extra memory is asymptotically relevant when the distance function is given implicitly (as in the case of Euclidean instances), it is irrelevant when there is an actual $O(mn)$ distance matrix (as in class RW). Given these considerations, we opted for using in this chapter the fastest version proposed by Resende and Werneck (2003), even though it requires $\Theta(mn)$ memory positions.

Since the implementation is rather intricate, we abstain from describing it here. The reader is referred to the original paper (Resende and Werneck, 2003) for details and for an experimental comparison with Whitaker's implementation.

11.5 INTENSIFICATION

In this section, we discuss the intensification aspects of our heuristic. We maintain a pool of *elite solutions*, high-quality solutions found during the execution. Intensification occurs in two different stages, as Figure 12.1 shows. First, every multistart iteration contains an intensification step, in which the newly generated solution is combined with a solution from the pool. Then, in the post-optimization phase, solutions in the pool are combined among themselves. In both stages, the strategy used to combine a pair of solutions is the same: *path-relinking*. Originally proposed for tabu search and scatter search (Glover, 1996; Glover et al., 2000), this procedure was first applied within the GRASP framework by Laguna and Martí (1999), and widely applied ever

since (Resende and Ribeiro (2003) present numerous examples). Subsection 11.5.1 briefly describes how path-relinking works. Subsection 11.5.2 explains the rules by which the pool is updated and solutions are taken from it. Finally, Subsection 11.5.3 discusses the post-optimization phase.

11.5.1 Path-relinking

Let S_1 and S_2 be two valid solutions, interpreted as sets of (open) facilities. The path-relinking procedure starts with one of the solutions (say, S_1) and gradually transforms it into the other (S_2) by swapping in elements from $S_2 \setminus S_1$ and swapping out elements from $S_1 \setminus S_2$. The total number of swaps made is $|S_2 \setminus S_1|$, which is equal to $|S_1 \setminus S_2|$; this value is known as the *symmetric difference* between S_1 and S_2. The choice of which swap to make in each stage is greedy: we always perform the most profitable (or least costly) move.

As pointed out by Resende and Ribeiro (2003), the outcome of the method is usually the best solution found in the path from S_1 to S_2. Here we use a slight variant: the outcome is the best *local minimum* in the path. A local minimum in this context is a solution that is both succeeded (immediately) and preceded (either immediately or through a series of same-value solutions) in the path by strictly worse solutions. If the path has no local minima, one of the original solutions (S_1 or S_2) is returned with equal probability. When there is an improving solution in the path, our criterion matches the traditional one exactly: it simply returns the best element in the path. It is different only when the stardard path-relinking is unsuccessful, in which case we try to increase diversity by selecting a solution other than the extremes of the path.

Note that path-relinking is very similar to the local search procedure described in Section 11.4, with two main differences. First, the number of allowed moves is restricted: only elements in $S_2 \setminus S_1$ can be inserted, and only those in $S_1 \setminus S_2$ can be removed. Second, non-improving moves are allowed. Fortunately, these differences are subtle enough to be incorporated into the basic implementation of the local search procedure. In fact, both procedures share much of their code in our implementation.

We further augment the intensification procedure by performing a full local search on the solution produced by path-relinking. Because this solution is usually very close to a local optimum, this application tends to be much faster than on a solution generated by the randomized constructive algorithm. A side effect of applying local search at this point is increased diversity, since we are free to use facilities that did not belong to any of the original solutions.

We note that this procedure has some similarity with VNS (Mladenović and Hansen, 1997). Starting from a local optimum, VNS obtains a solution in some extended neighborhood and applies local search to it, hoping to find a better solution. The main difference is that VNS uses a randomized method to find the neighboring solution, while we use a second local optimum as a guide. The distance from the new solution to the original one (actually, to both extremes) is at least two in our case.

11.5.2 Pool Management

An important aspect of the algorithm is managing the pool of elite solutions. Empirically, we observed that an application of path-relinking to a pair of solutions is less likely to be successful if the solutions are very similar. The longer the path between the solutions, the greater the probability that an entirely different local minimum (as opposed to the original solutions themselves) will be found. It is therefore reasonable to take into account not only solution quality, but also diversity when dealing with the pool of elite solutions.

The pool must support two essential operations: insertion of new solutions (represented by the `add` function in Figure 12.1) and selection of a solution for path-relinking (the `select` function in the pseudocode). We describe each of these in turn.

11.5.2.1 Insertion. For a solution S with cost $c(S)$ to be added to the pool, two conditions must be met. First, its symmetric difference from all solutions in the pool whose value is less than $c(S)$ must be at least four; after all, path-relinking between solutions that differ by fewer than four facilities cannot produce solutions that are better than both original extremes, since they are local optima. Second, if the pool is full, the solution must be at least as good as the worst elite solution (if the pool is not full, this is obviously not necessary).

If both conditions are met, the solution is inserted. If the pool is not full and the new solution is not within distance four of any other elite solution (including worse ones), it is simply added. Otherwise, it replaces the most similar solution among those of equal or higher value.

11.5.2.2 Selection. In every iteration of the algorithm, a solution is selected from the pool (Figure 12.1, line 6) and combined with S, the solution most recently found. An approach that has been applied to other problems with some degree of success is to select a solution uniformly at random (Resende and Ribeiro, 2003). However, this often means selecting a solution that is too similar to S, thus making the procedure unlikely to find good new solutions. To minimize this problem, we pick solution from the pool with probabilities proportional to their symmetric difference with respect to S. In Section 11.5.4.3, we show empirical evidence that this strategy does pay off.

11.5.3 Post-optimization

In the process of looking for a good solution, the multistart phase of our heuristic produces not one, but several different local optima, which are often not much worse than the best solution found. The *post-optimization phase* in our algorithm combines these solutions to obtain even better ones. This phase takes as input the pool of elite solutions, whose construction was described in previous sections. Every solution in the pool is combined with each other by path-relinking. The solutions generated by this process are added to a new pool of elite solutions (following the constraints described in Section 11.5.2), representing a new *generation*. The algorithm proceeds until it creates a generation that does not improve upon previous generations. Recently, sim-

ilar multi-generation path-relinking strategies have been used successfully within the GRASP framework (Aiex et al., 2003; Ribeiro et al., 2002). The generic idea of combining solutions to obtain new ones is not new, however; it is one of the basic features of genetic algorithms (Goldberg, 1989; Michalewicz, 1994).

11.5.4 Empirical Analysis

In this section, we analyze empirically some aspects of the intensification strategy. First, in Section 11.5.4.1, we show how the execution of path-relinking during the multistart phase (and not only during post-optimization) helps the algorithm find good solutions faster. Then, in Section 11.5.4.2, we examine the question of which *direction* to choose when performing path-relinking between two solutions S_1 or S_2: from S_1 to S_2, from S_2 to S_1, or both? Finally, Section 11.5.4.3 compares different strategies for selecting solutions from the pool in the multistart phase.

11.5.4.1 Path-relinking in the Multistart Phase. Our implementation is such that the randomized constructive solution produced in each multistart iteration depends only on the initial seed, regardless of whether path-relinking is executed or not. Therefore, if the number of iterations is the same, the addition of path-relinking to the multistart phase cannot decrease solution quality. It could be the case, however, that the extra time spent on path-relinking would lead to even better results if used for additional iterations instead.

To test this hypothesis, we took a few representative instances and ran both versions of the heuristic (with and without path-relinking) for a period 100 times as long as the average time it takes to execute one iteration (construction followed by local search) *without* path-relinking. We then compared the quality of the solutions obtained as the algorithm progressed. The constructive algorithm used was sample. Results in this test do not include post-optimization. We selected one instance from each class (fl1400 from TSP, pmed40 from ORLIB, and rw500 from RW), and tested each with seven values of p, from 10 to roughly one third of the number of facilities (m). The test was repeated 10 times for each value of p, with 10 different seeds.

Figure 11.2 refers fl1400 with $p = 500$. The graph shows how solution quality improves over time. Both quality and time are normalized. Times are given in multiples of the average time it takes to perform one multistart iteration without path-relinking (this average is taken over all iterations of all 10 runs).[5] Solution quality is given as a fraction of the average solution value found by the first iteration (again, without path-relinking).

Figures 11.3, 11.4 and 11.5 refer to the same experiment. Each curve in those graphs represents an instance with a particular value of p. Times are normalized as before. The *quality ratio*, shown in the vertical axis, is the ratio between the average solution qualities obtained with and without path-relinking. Values smaller than 1.000 favor path-relinking.

[5]Note that the first time value shown in the graph is 2; at time 1, not all ratios are defined because in some cases the first iteration takes more than average time to execute.

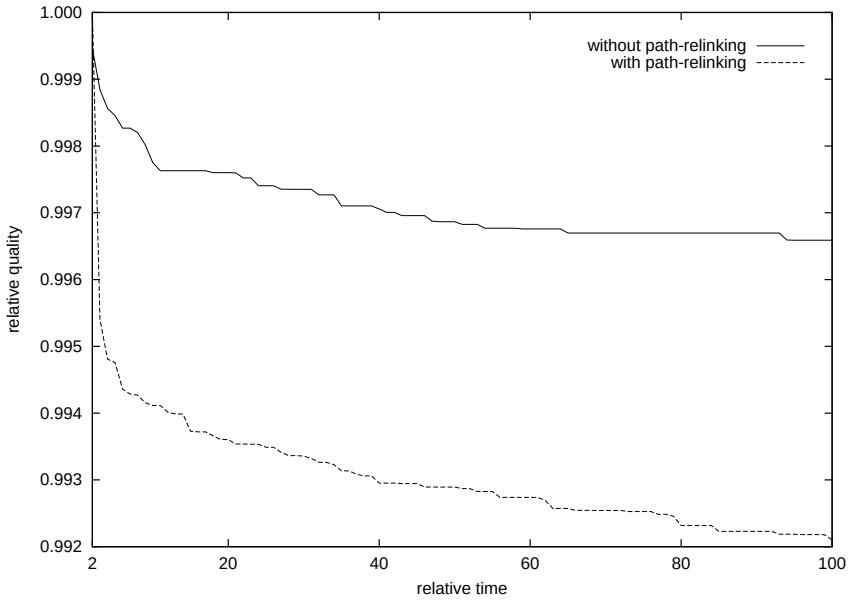


Figure 11.2 Instance fl1400, $p = 500$: Quality of the best solution found as a fraction of the average value of the first solution. Times are given as multiples of the average time required to perform one multistart iteration. Smaller values are better.

These results confirm what should be expected. If very few iterations are performed, path-relinking is not particularly helpful; solutions of comparable quality (or even better) can be found using a "pure" multistart approach (construction followed by local search). However, if more time is to be spent, using path-relinking is a good strategy, consistently leading to solutions of superior quality within the same time frame. This is especially true for harder instances, those in which p is large. Instance rw500 seems to be an exception; as p becomes greater than 75, the problem apparently becomes easier.

11.5.4.2 Direction. An important aspect of path-relinking is the *direction* in which it is performed. Given two solutions S_1 and S_2, we must decide whether to go from S_1 to S_2, from S_2 to S_1, or both. We tested the following criteria:

- random: Direction picked uniformly at random.
- up: From the best to the worst solution among the two; this has the potential advantage of exploring more carefully the most promising vicinity.
- down: From the worst to the best solution; by exploring more carefully the vicinity of the worst solution, it can find good solutions that are relatively far from the best known solutions, thus favoring diversity.

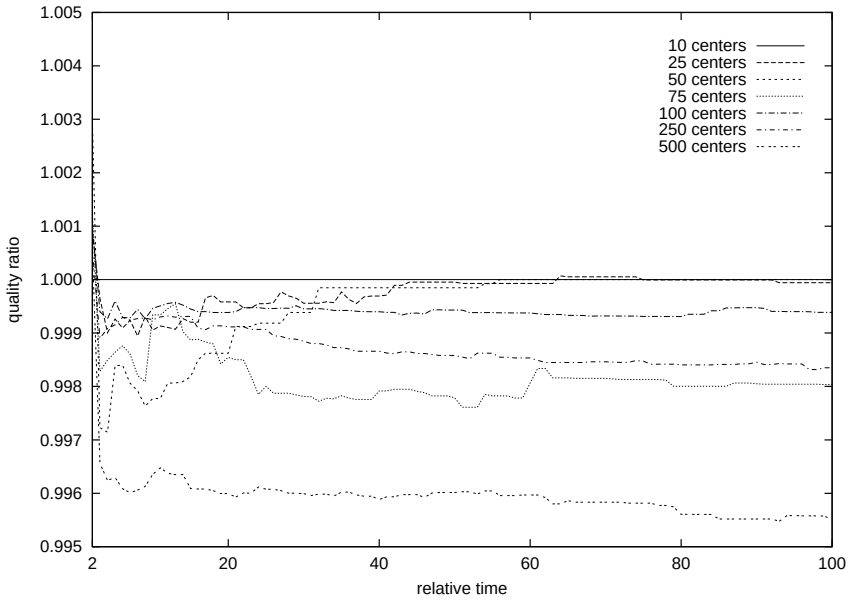


Figure 11.3 Instance fl1400 (class TSP): Ratios between partial solutions found with and without path-relinking. Times are normalized with respect to the average time it takes to execute one multistart iteration. Values smaller than 1.000 favor the use of path-relinking.

- new: Start from the newly generated solution, not from the one already in the pool (this strategy applies only to the multistart phase of the algorithm, not to the post-optimization stage). Again, the goal is to obtain greater solution diversity.
- none: Do not perform path-relinking during the multistart phase (this strategy cannot be applied in the post-optimization stage).
- both: Perform path-relinking in both directions and return the best result. This method is guaranteed to find the best solution in each case, but it takes roughly twice as much time as the other methods.

We tested all valid combinations of these methods on the 10 instances of the restricted set defined in Section 11.2.1, each with 10 different seeds. We ran our algorithm with 32 iterations and 10 elite solutions, using sample as the constructive method. Tables 11.2, 11.3, and 11.4 show the results obtained in the experiment. (The definitions of *average relative percentage deviation* and *normalized relative rank*, used in these tables, are given in Section 11.3.)

Note that some strategies can be discarded for being too slow without any clear improvement in solution quality. That is the case of those that use strategy both in the post-optimization phase (and also during the first stage of the algorithm, although the extra time in this case is far less relevant). Furthermore, using path-relinking during

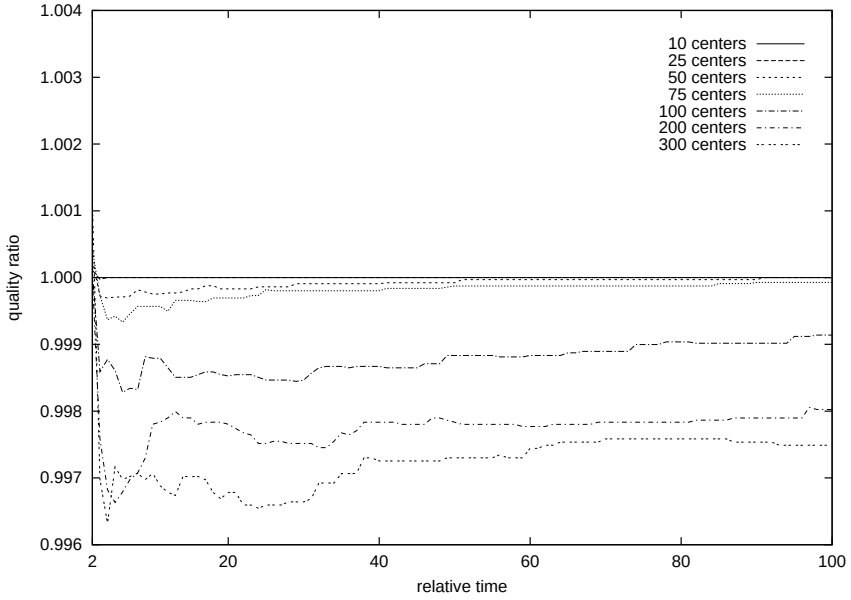


Figure 11.4 Instance pmed40 (class ORLIB): Ratios between partial solutions found with and without path-relinking. Times are normalized with respect to the average time it takes to execute one multistart iteration. Values smaller than 1.000 favor the use of path-relinking.

Table 11.2 Solution quality of HYBRID with different path-relinking strategies: Average relative percentage deviations. Each value represents how much the average solution value found by each method is above (or below) the average found by all methods. Smaller values are better.

MULTISTART METHOD	POST-OPTIMIZATION METHOD			
	both	down	random	up
none	0.056	0.056	0.033	0.024
both	0.005	0.009	-0.030	-0.007
down	-0.010	0.007	-0.009	-0.012
random	0.001	0.004	-0.002	0.001
new	-0.008	0.004	-0.007	-0.011
up	-0.029	-0.032	-0.019	-0.022

the multistart stage is clearly important; even though it is still possible to obtain above-average solutions eventually if none is used in that phase, this only happens if both is the strategy used in post-optimization — which results in much longer running times.

Among the remaining strategies, Tables 11.2 and 11.3 show no clearly dominant one. Several combinations of new, up, down, and random seem like reasonable choices. Five have better-than-average quality according to both measures used: up:down, down:random, random:random, new:random, and up:random (in our notation, the first

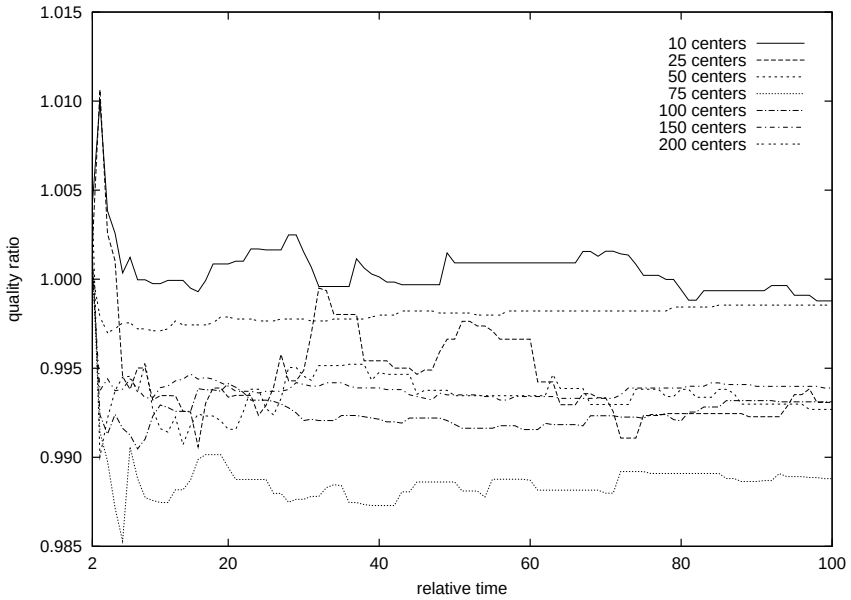


Figure 11.5 Instance rw500 (class RW): Ratios between partial solutions found with and without path-relinking. Times are normalized with respect to the average time it takes to execute one multistart iteration. Values smaller than 1.000 favor the use of path-relinking.

Table 11.3 Solution quality of HYBRID with different path-relinking strategies: Average normalized ranks. Smaller values are better.

MULTISTART METHOD	POST-OPTIMIZATION METHOD			
	both	down	random	up
none	-0.017	0.565	0.448	0.465
both	-0.117	-0.143	-0.270	0.174
down	-0.357	0.270	-0.265	0.004
random	-0.183	0.209	-0.100	0.161
new	-0.387	-0.030	-0.135	0.078
up	-0.283	-0.209	-0.061	0.183

Table 11.4 HYBRID running times with different path-relinking strategies: Average relative percent deviation with respect to the average.

MULTISTART METHOD	POST-OPTIMIZATION METHOD			
	both	down	random	up
none	33.3	-12.2	-7.3	-6.9
both	27.8	-2.3	-0.5	-2.3
down	22.7	-9.4	-7.8	-9.4
random	20.1	-12.0	-9.7	-10.9
new	20.3	-8.7	-8.0	-11.1
up	23.1	-9.3	-10.0	-9.6

method refers to the multistart phase of the algorithm, the second to the post-optimization stage). We decided to use up:down in the final version of our algorithm, since this was the method with the best average relative percentage deviation and a good average rank. This method has the interesting feature of favoring quality when dealing with lower-quality solutions (during the multistart phase), and diversity when the overall solution quality is higher (during the post-optimization phase).

11.5.4.3 Selection Strategy. We have shown that applying path-relinking during the first stage of the algorithm helps finding good solutions faster. Here, we analyze the criterion used to choose the elite solution to be combined with S, the solution obtained after local search. Recall that the usual method is to select the solution uniformly at random, and that we propose picking solutions with probabilities proportional to their symmetric difference with respect to S. We call these strategies uniform and biased, respectively.

When performing path-relinking between a pair of solutions, our goal is to obtain a third solution of lower cost. We consider the combination *successful* when this happens. The ultimate goal of the selection scheme is to find, among the elite solutions, one that leads to a successful combination. Better selection schemes will find one such solution with higher probability.

To determine which method is better according to this criterion, we performed the following experiment on each of the 10 instances in the restricted set defined in Section 11.2.1. First, run the multistart heuristic (without path-relinking) until a pool of 110 solutions is filled. Then, take the top 10 solutions (call them $E_1, E_2, \ldots, E_{10}$) obtained and create a new pool. Denote the remaining 100 solutions by $S_1, S_2, \ldots, S_{100}$. Perform path-relinking between each of these 100 solutions and each solution in the pool, and decide based on the results which selection method (biased or uniform) would have a greater probability of success if we had to select one of the 10 instances.

To compute the probability of success of each method, we need some definitions. Let $s(i, j)$ be 1 if the path-relinking between S_i and E_j is successful, and 0 otherwise; also, let $\Delta(i, j)$ be the symmetric difference between S_i and E_j. For a given solution S_i, the probability of success for uniform, if it were applied, would be

$$u_i = \frac{\sum_{j=1}^{10} s(i, j)}{10}.$$

On the other hand, the probability of success of biased would be

$$b_i = \frac{\sum_{j=1}^{10} [s(i, j) \cdot \Delta(i, j)]}{\sum_{i=1}^{10} \Delta(i, j)}.$$

For each of the 10 instances, the procedure described above was executed 10 times, with 10 seeds, always using sample as the constructive algorithm and up as the path-relinking direction. Therefore, for each instance, 1,000 selections were simulated (100 for each random seed).

The results are summarized in Table 11.5. For each instance, we show the percentage of cases in which one method has greater probability of success than the other (when the probabilities are equal, we consider the experiment a tie).

Table 11.5 Comparison between the uniform and biased selection schemes. Values represent percentage of cases in which one method has greater probability of leading to a successful relink than the other.

INSTANCE		SELECTION METHOD		
NAME	p	uniform	TIE	biased
fl1400	150	38.7	10.8	50.5
fl1400	500	0.0	99.9	0.1
gr150	25	34.9	5.6	59.5
pcb3038	30	45.2	5.4	49.4
pcb3038	250	0.2	98.5	1.3
pmed15	100	14.5	4.5	81.0
pmed40	90	14.2	5.2	80.6
rw500	25	39.8	10.3	49.9
rw500	75	32.0	11.3	56.7
sl700	233	6.4	56.2	37.4

Note that in all cases biased has superior performance, sometimes by a significant margin. In two cases the probability of a tie was almost 100%; this is due to the fact that path-relinking almost always works for those particular instances — any selection scheme would be successful. In situations where there were "wrong" alternatives, biased was better at avoiding them.

11.6 FINAL RESULTS

This section presents detailed results obtained by the final version of our algorithm, built based on the experiments reported in previous sections. It uses sample as the randomized constructive heuristic (see Section 11.3); path-relinking is executed in both stages of the algorithm (Section 11.5.4.1): from the best to the worst solution during the multistart phase, and from the worst to the best during post-optimization (Section 11.5.4.2); and solutions are selected from the pool in a biased way during the multistart phase (Section 11.5.4.3). The results reported here refer to runs with 32 multistart iterations and 10 elite solutions — of course, these numbers can be changed to make the algorithm faster (if they are reduced) or to obtain better solutions (if they are increased).

We tested our algorithm on all instances mentioned in Section 11.2.1. We ran it nine times on each instance, with different seeds. Tables 11.6 to 11.12 present the results. The last three columns refer to the full version of our method, whereas the three that immediately precede them refer to the multistart phase only. In each case, we present three different values: first, the *median* value obtained (which always corresponds to some valid solution to the problem); second, the *average percentage error* (%ERR), which indicates how much the average value obtained by our method is above the best solution known (in percentage terms); third, the average running time in seconds. All three measures consider the nine runs of the algorithm.

For reference, the tables also contain the lowest (to the best of our knowledge) upper bounds on solution values available in the literature at the time of writing for

each of the instances tested. The optimum values are known for all instances in three classes: ORLIB (Beasley, 1985), SL (Senne and Lorena, 2000), and GR (Senne and Lorena, 2000). For class TSP, we list the best upper bounds in the literature, as well as references to the papers that first presented the bounds shown (they are presented in the SOURCE column in Tables 11.6, 11.7, and 11.8). The bounds do not necessarily correspond to solutions found by the main heuristics described in those papers — in some cases, they were found by other, more time-consuming methods. For several instances, in at least one of the nine runs our procedure was able to improve the best bound known. When that was the case, the improved bound is presented, and the SOURCE column contains a dash (—). These values should not be considered the "final results" of our method when compared to others, since they refer to especially successful runs; the truly representative results are the medians and averages listed in the tables. Because class RW was introduced only recently (Resende and Werneck, 2003), no good upper bounds were available. Therefore, the BEST column in Table 11.12 presents the best solution found by the nine runs of our algorithm in each case.

Table 11.6 Final results for fl1400, an Euclidean instance from class TSP with 1400 nodes: median values, average percentage errors, and running times in seconds. Best results reported by Hansen and Mladenović (1997) and by Hansen et al. (2001) are denoted by HMP97 and HMP01, respectively. All other best values were found by HYBRID itself.

	BEST KNOWN		SINGLE-STAGE HYBRID			DOUBLE-STAGE HYBRID		
p	VALUE	SOURCE	MED	%ERR	TIME	MED	%ERR	TIME
10	101249.47	HMP01	101249.55	0.000	117.1	101249.55	0.000	118.5
20	57857.55	HMP01	57857.94	0.001	76.8	57857.94	0.001	83.5
30	44013.48	—	44013.48	0.003	76.0	44013.48	0.000	106.2
40	35002.52	—	35002.60	0.007	68.6	35002.60	0.003	101.3
50	29089.78	HMP01	29090.23	0.002	58.8	29090.23	0.002	73.9
60	25161.12	—	25166.91	0.028	57.0	25164.02	0.012	91.5
70	22125.53	HMP01	22126.03	0.006	50.6	22126.03	0.002	70.2
80	19872.72	—	19878.45	0.046	49.8	19876.57	0.018	78.1
90	17987.94	HMP01	18006.83	0.091	48.7	17988.60	0.013	74.2
100	16551.20	HM97	16567.01	0.099	47.3	16559.82	0.051	82.4
150	12026.47	—	12059.12	0.264	48.7	12036.00	0.068	132.5
200	9359.15	—	9367.98	0.098	49.4	9360.67	0.017	101.3
250	7741.51	—	7754.50	0.165	54.5	7746.31	0.057	130.3
300	6620.92	—	6637.81	0.258	57.8	6623.98	0.041	167.1
350	5720.91	—	5749.51	0.489	59.6	5727.17	0.097	177.6
400	5006.83	—	5033.96	0.571	64.1	5010.22	0.087	157.5
450	4474.96	—	4485.16	0.226	68.3	4476.68	0.059	170.7
500	4047.90	—	4059.16	0.265	71.9	4049.56	0.044	210.9

The tables show that our method found solutions within at most 0.1% of the previous best known solutions in all cases. The only exception is class RW, for which there were greater deviations. Although in these cases they were computed with respect to solutions found by HYBRID itself, this does suggest that our method obtains better

Table 11.7 Final results for pcb3038, an Euclidean instance from class TSP with 3038 nodes: median values, average percentage errors, and running times in seconds. Best results reported by Hansen et al. (2001) and by Taillard (2003) are denoted by HMP01 and Tai03, respectively. All other best values were found by HYBRID itself.

	BEST KNOWN		SINGLE-STAGE HYBRID			DOUBLE-STAGE HYBRID		
p	VALUE	SOURCE	MED	%ERR	TIME	MED	%ERR	TIME
10	1213082.03	—	1213082.03	0.000	1115.8	1213082.03	0.000	1806.3
20	840844.53	—	840844.53	0.004	647.9	840844.53	0.003	943.4
30	677306.76	—	678108.52	0.111	426.7	677436.66	0.038	847.0
40	571887.75	—	572012.44	0.054	312.6	571887.75	0.004	492.6
50	507582.13	—	507754.72	0.050	251.7	507663.80	0.013	472.4
60	460771.87	—	461194.61	0.102	218.2	460797.55	0.024	481.4
70	426068.24	—	426933.75	0.198	201.3	426153.31	0.020	470.9
80	397529.25	—	398405.57	0.234	188.5	397585.89	0.018	555.9
90	373248.08	—	374152.75	0.259	182.3	373488.82	0.061	380.8
100	352628.35	—	353576.86	0.289	174.0	352755.13	0.033	448.1
150	281193.96	Tai03	282044.70	0.297	163.3	281316.82	0.041	402.5
200	238373.26	—	238984.42	0.267	162.0	238428.35	0.030	406.9
250	209241.25	Tai03	209699.36	0.204	171.8	209326.83	0.041	407.5
300	187712.12	—	188168.32	0.223	184.4	187763.64	0.029	395.8
350	170973.34	Tai03	171443.87	0.266	200.0	171048.03	0.035	412.0
400	157030.46	Tai03	157414.79	0.251	203.4	157073.20	0.029	436.3
450	145384.18	—	145694.26	0.212	216.3	145419.81	0.023	462.3
500	135467.85	Tai03	135797.08	0.257	231.1	135507.73	0.030	478.5
550	126863.30	—	127207.83	0.267	243.8	126889.89	0.025	514.0
600	119107.99	HMP01	119428.60	0.266	258.3	119135.62	0.026	595.8
650	112063.73	—	112456.15	0.339	271.0	112074.74	0.013	619.0
700	105854.40	—	106248.00	0.360	284.0	105889.22	0.034	637.3
750	100362.55	HMP01	100713.79	0.337	296.4	100391.53	0.034	649.3
800	95411.78	—	95723.00	0.317	286.6	95432.66	0.023	677.8
850	91003.62	—	91268.56	0.298	296.1	91033.10	0.030	689.3
900	86984.10	—	87259.78	0.302	306.4	87022.59	0.037	730.4
950	83278.78	—	83509.58	0.265	314.3	83299.22	0.023	780.5
1000	79858.79	—	80018.33	0.193	321.7	79869.98	0.013	806.2

results in absolute terms on instances with well-defined metrics (graphs and Euclidean instances), than on random instances (such as class RW).

11.6.1 Other Methods

We now analyze how our algorithm behaves in comparison with other methods in the literature. We refer to our method (including the post-optimization phase) as HYBRID. For reference, we also present the results obtained only by the multistart phase of the algorithm, called HYB-SS (for "hybrid, single-stage"). The results presented in this section are averages taken from the %ERR and TIME columns from Tables 11.6 to 11.12.

Other methods considered in the comparison are:

Table 11.8 Final results for instance rl5934, an Euclidean instance from class TSP with 5934 nodes: median values, average percentage errors, and running times in seconds. Best results reported by Hansen et al. (2001) are denoted by HMP01. All other best values were found by HYBRID itself.

	BEST KNOWN		SINGLE-STAGE HYBRID			DOUBLE-STAGE HYBRID		
p	VALUE	SOURCE	MED	%ERR	TIME	MED	%ERR	TIME
10	9794951.00	HMP01	9794973.65	0.000	5971.1	9794973.65	0.000	8687.1
20	6718848.19	—	6719116.39	0.007	3296.8	6719026.03	0.003	4779.6
30	5374936.14	—	5379979.09	0.131	2049.8	5376040.45	0.017	4515.1
40	4550364.60	—	4550843.75	0.022	1470.4	4550518.95	0.004	2499.3
50	4032379.97	—	4033758.13	0.059	1195.3	4032675.94	0.014	2280.6
60	3642397.88	—	3646198.03	0.089	996.1	3642949.30	0.022	2244.0
70	3343712.45	—	3348834.92	0.164	872.5	3344888.24	0.039	2138.3
80	3094824.49	—	3099917.93	0.150	778.8	3095442.55	0.033	1792.4
90	2893362.39	—	2898721.66	0.169	708.8	2894954.78	0.050	1844.2
100	2725180.81	—	2730313.90	0.180	671.2	2725580.72	0.015	1892.6
150	2147881.53	—	2151985.53	0.182	560.2	2148749.47	0.035	1209.2
200	1808179.07	—	1812249.63	0.209	526.6	1808658.73	0.029	1253.0
250	1569941.34	—	1573800.83	0.229	526.2	1570445.77	0.037	1203.8
300	1394115.39	—	1397064.23	0.229	550.1	1394361.41	0.022	1042.7
350	1256844.04	—	1259733.85	0.226	575.6	1257098.17	0.027	1246.4
400	1145669.38	HMP01	1148386.49	0.224	583.8	1145961.13	0.033	1157.6
450	1053363.64	—	1055756.67	0.226	619.2	1053729.79	0.040	1236.9
500	973995.18	—	975940.78	0.197	641.7	974242.08	0.027	1236.7
600	848283.85	—	849765.46	0.174	703.7	848499.21	0.021	1439.4
700	752068.38	HMP01	753522.21	0.189	767.3	752263.82	0.028	1566.6
800	676795.78	—	678300.99	0.205	782.1	676956.64	0.027	1574.9
900	613367.44	HMP01	614506.49	0.183	834.5	613498.64	0.024	1722.0
1000	558802.38	HMP01	559797.83	0.178	877.7	558943.93	0.024	1705.3
1100	511813.19	HMP01	512793.56	0.203	931.4	511928.86	0.026	1893.4
1200	470295.38	HMP01	471486.76	0.249	988.1	470411.12	0.023	2082.0
1300	433597.44	HMP01	434688.75	0.258	1033.4	433678.02	0.020	2147.8
1400	401853.00	HMP01	402796.80	0.232	1072.4	401934.24	0.020	2288.7
1500	374014.57	—	374803.24	0.207	1029.7	374056.40	0.012	2230.3

- VNS: Variable Neighborhood Search, by Hansen and Mladenović (1997). Results for this method are available for the ORLIB class (all 40 instances were tested, with running times given for only 22 of them), for fl1400 (all 18 values of p), and pcb3038 (with only 10 values of p: $50, 100, 150, \ldots, 500$). The values shown here were computed from those reported in Tables 1, 2, and 3 of Hansen and Mladenović (1997).

- VNDS: Variable Neighborhood Decomposition Search, by Hansen et al. (2001). Results are available for all ORLIB and TSP instances.[6]

[6]The authors also tested instances from Rolland et al. (1996); unfortunately, we were unable to obtain these instances at the time of writing.

Table 11.9 Final results obtained for class ORLIB, graph-based instances introduced by Beasley (1985): median values, average percentage errors, and running times in seconds.

	INSTANCE			SINGLE-STAGE HYBRID			DOUBLE-STAGE HYBRID		
NAME	n	p	OPT	MED	%ERR	TIME	MED	%ERR	TIME
pmed01	100	5	5819	5819	0.000	0.5	5819	0.000	0.5
pmed02	100	10	4093	4093	0.000	0.4	4093	0.000	0.5
pmed03	100	10	4250	4250	0.000	0.4	4250	0.000	0.5
pmed04	100	20	3034	3034	0.000	0.4	3034	0.000	0.5
pmed05	100	33	1355	1355	0.000	0.4	1355	0.000	0.5
pmed06	200	5	7824	7824	0.000	1.8	7824	0.000	1.8
pmed07	200	10	5631	5631	0.000	1.4	5631	0.000	1.4
pmed08	200	20	4445	4445	0.000	1.2	4445	0.000	1.2
pmed09	200	40	2734	2734	0.000	1.2	2734	0.000	1.5
pmed10	200	67	1255	1255	0.000	1.3	1255	0.000	1.6
pmed11	300	5	7696	7696	0.000	3.5	7696	0.000	3.5
pmed12	300	10	6634	6634	0.000	2.9	6634	0.000	2.9
pmed13	300	30	4374	4374	0.000	2.4	4374	0.000	2.5
pmed14	300	60	2968	2968	0.000	2.9	2968	0.000	3.5
pmed15	300	100	1729	1729	0.013	3.3	1729	0.006	4.3
pmed16	400	5	8162	8162	0.000	8.1	8162	0.000	8.2
pmed17	400	10	6999	6999	0.000	6.1	6999	0.000	6.3
pmed18	400	40	4809	4809	0.005	5.5	4809	0.005	6.7
pmed19	400	80	2845	2845	0.000	6.3	2845	0.000	7.5
pmed20	400	133	1789	1789	0.000	7.1	1789	0.000	8.6
pmed21	500	5	9138	9138	0.000	12.2	9138	0.000	12.2
pmed22	500	10	8579	8579	0.000	10.7	8579	0.000	11.3
pmed23	500	50	4619	4619	0.000	9.4	4619	0.000	11.0
pmed24	500	100	2961	2961	0.000	11.4	2961	0.000	13.1
pmed25	500	167	1828	1828	0.006	13.4	1828	0.000	16.2
pmed26	600	5	9917	9917	0.000	20.5	9917	0.000	20.5
pmed27	600	10	8307	8307	0.000	16.4	8307	0.000	16.4
pmed28	600	60	4498	4498	0.005	14.6	4498	0.000	17.4
pmed29	600	120	3033	3033	0.000	18.0	3033	0.000	21.0
pmed30	600	200	1989	1989	0.028	21.1	1989	0.000	26.9
pmed31	700	5	10086	10086	0.000	28.8	10086	0.000	28.8
pmed32	700	10	9297	9297	0.000	22.8	9297	0.000	22.9
pmed33	700	70	4700	4700	0.000	20.6	4700	0.000	23.7
pmed34	700	140	3013	3013	0.011	25.8	3013	0.000	30.8
pmed35	800	5	10400	10400	0.000	36.7	10400	0.000	36.7
pmed36	800	10	9934	9934	0.000	31.7	9934	0.000	34.4
pmed37	800	80	5057	5057	0.000	28.8	5057	0.000	32.4
pmed38	900	5	11060	11060	0.000	52.9	11060	0.000	52.9
pmed39	900	10	9423	9423	0.000	36.5	9423	0.000	36.5
pmed40	900	90	5128	5129	0.020	36.6	5128	0.011	43.4

- LOPT: Local Optimization method, proposed by Taillard (2003). The method works by heuristically solving locally defined subproblems and integrating them into a solution to the main problem. The author provides detailed results (in Table 7) only for instance pcb3038, with nine values of p, all multiples of 50 between 100 to 500.

Table 11.10 Final results for class SL, graph-based instances introduced by Senne and Lorena (2000): median values, average percentage errors, and running times in seconds.

INSTANCE				SINGLE-STAGE HYBRID			DOUBLE-STAGE HYBRID		
NAME	n	p	OPT	MED	%ERR	TIME	MED	%ERR	TIME
sl700	700	233	1847	1848	0.060	30.2	1847	0.000	39.5
sl800	800	267	2026	2027	0.033	41.8	2026	0.000	53.2
sl900	900	300	2106	2107	0.037	54.1	2106	0.011	68.2

Table 11.11 Final results for class GR, graph-based instances introduced by Galvão and ReVelle (1996): median values, average percentage errors, and running times in seconds.

INSTANCE			SINGLE-STAGE HYBRID			DOUBLE-STAGE HYBRID		
NAME	p	OPT	MED	%ERR	TIME	MED	%ERR	TIME
gr100	5	5703	5703	0.000	0.5	5703	0.000	0.5
	10	4426	4426	0.105	0.6	4426	0.070	1.0
	15	3893	3893	0.000	0.5	3893	0.000	0.8
	20	3565	3565	0.009	0.4	3565	0.000	0.7
	25	3291	3291	0.003	0.4	3291	0.000	0.7
	30	3032	3032	0.000	0.4	3032	0.000	0.6
	40	2542	2542	0.000	0.4	2542	0.000	0.6
	50	2083	2083	0.011	0.4	2083	0.005	0.6
gr150	5	10839	10839	0.000	1.3	10839	0.000	1.3
	10	8729	8729	0.033	1.1	8729	0.017	2.0
	15	7390	7390	0.036	1.0	7390	0.011	1.7
	20	6454	6462	0.167	0.9	6462	0.083	1.5
	25	5875	5887	0.246	0.9	5875	0.100	1.7
	30	5495	5502	0.135	0.8	5495	0.010	1.5
	40	4907	4907	0.011	0.8	4907	0.002	1.2
	50	4374	4375	0.025	0.8	4375	0.015	1.2

- DEC: Decomposition Procedure, also studied by Taillard (2003) and based on the decomposition of the original problem. Results are provided for the same nine instances as LOPT.
- LSH: Lagrangean-Surrogate Heuristic, described by Senne and Lorena (2000). Their paper contains results for six ORLIB instances (pmed05, pmed10, pmed15, pmed20, pmed25, pmed30), for nine values of p for pcb3038 (the same nine used with LOPT), and for all instances in classes SL and GR. Our comparison uses values taken from Tables 1, 2, and 3 in the paper.
- CGLS: Column Generation with Lagrangean/Surrogate Relaxation, studied by Senne and Lorena (2002). Results are available for 15 ORLIB instances (pmed01, pmed05, pmed06, pmed07, pmed10, pmed11, pmed12, pmed13, pmed15, pmed16, pmed17, pmed18, pmed20, pmed25, and pmed30), for all three SL instances, and for five values of p on instance pcb3038 (300, 350, 400, 450, and 500). We

Table 11.12 Final results for class RW, random instances introduced by Resende and Werneck (2003): median values, average percentage errors, and running times in seconds.

INSTANCE			SINGLE-STAGE HYBRID			DOUBLE-STAGE HYBRID		
NAME	p	BEST	MED	%ERR	TIME	MED	%ERR	TIME
rw100	10	530	530	0.042	0.7	530	0.000	1.3
	20	277	277	0.000	0.5	277	0.000	0.7
	30	213	213	0.000	0.4	213	0.000	0.5
	40	187	187	0.000	0.3	187	0.000	0.5
	50	172	172	0.000	0.3	172	0.000	0.4
rw250	10	3691	3691	0.084	6.1	3691	0.063	10.4
	25	1364	1370	0.587	3.3	1364	0.204	5.8
	50	713	718	0.701	2.1	713	0.109	3.9
	75	523	523	0.064	1.9	523	0.000	2.6
	100	444	444	0.000	1.8	444	0.000	2.2
	125	411	411	0.000	1.5	411	0.000	2.0
rw500	10	16108	16259	0.813	33.1	16108	0.068	76.9
	25	5681	5749	0.974	20.8	5683	0.241	46.9
	50	2628	2657	1.120	14.1	2635	0.364	27.7
	75	1757	1767	0.746	11.6	1757	0.177	20.5
	100	1380	1388	0.515	11.5	1382	0.105	20.4
	150	1024	1026	0.174	11.1	1024	0.011	15.4
	200	893	893	0.025	11.8	893	0.000	14.4
	250	833	833	0.000	9.6	833	0.000	11.6
rw1000	10	67811	68202	0.642	153.6	68136	0.466	256.3
	25	24896	25192	1.375	111.1	24964	0.451	293.5
	50	11306	11486	1.501	77.7	11360	0.602	169.1
	75	7161	7302	1.930	60.2	7207	0.576	160.1
	100	5223	5297	1.500	55.5	5259	0.598	109.8
	200	2706	2727	0.756	57.5	2710	0.164	100.4
	300	2018	2021	0.099	55.2	2018	0.022	71.5
	400	1734	1734	0.013	61.8	1734	0.000	73.5
	500	1614	1614	0.000	47.9	1614	0.000	55.9

consider here the results found by method *CG(t)*, taken from Tables 1, 2, and 4 in the paper.

Table 11.13 presents, for each of the methods studied, the average percentage deviation with respect to the best solutions known, as given by Tables 11.6 to 11.11 above. Values for HYBRID and HYB-SS were computed from the %ERR columns in those tables. Each instance in class TSP is shown separately to allow a more precise analysis of the algorithms. Values in *slanted font* indicate that not all instances in the set were considered in the paper describing the method. A dash (—) is shown when no result for the class is available. Class RW is not included in this comparison, since the only results available are those obtained by our method.

The table shows that HYBRID is the only one whose average results are within 0.04% of the best values known for all classes. Furthermore, it obtained the best results on average in five out of six sets of instances. The only exception is class ORLIB: LSH found the optima of the six instances on which it was tested, whereas our

Table 11.13 Average percentage deviations of each method with respect to the best solution known. Values in *slanted font* indicate that not all instances in the set were tested by the method. Smaller values are better.

SERIES	HYBRID	HYB-SS	CGLS	DEC	LOPT	LSH	VNDS	VNS
GR	0.020	0.049	—	—	—	0.727	—	—
SL	0.004	0.043	0.691	—	—	0.332	—	—
ORLIB	0.001	0.002	*0.101*	—	—	*0.000*	0.116	0.007
fl1400	0.032	0.145	—	—	—	—	0.071	0.191
pcb3038	0.026	0.222	*0.043*	*4.120*	*0.712*	*2.316*	0.117	*0.354*
rl5934	0.024	0.170	—	—	—	—	0.142	—

method remained within 0.001% of optimality on all 40 instances (if we consider the median value obtained by HYBRID on each instance, instead of the average, it does find all 40 optima).

In any case, the difference between HYBRID and other methods is often very small. Several methods are virtually as good as ours in one or another class: that is the case of VNDS for all three TSP instances; of VNS and LSH for ORLIB instances; and of CGLS for pcb3038. This reveals the greatest strength of our method: *robustness*. It was able to obtain competitive results for all classes of instances. No other method among those tested has shown such degree of consistency.

Of course, we also have to consider the running times of the methods involved. Since we do not have access to all the algorithms compared, we present the running times reported by their authors. However, because different machines were used in each case, a direct comparison is impossible. For reference, Table 11.14 presents rough estimates of the relative speed of the machines involved. It shows the number of megaflops per second as reported by Dongarra (2003). These values refer to the number of floating-point operations — not terribly relevant for most algorithms compared, but they at least give an idea of the relative performance of the machines. Whenever the exact model reported in a paper (shown in the second column of Table 11.14) was not in Dongarra's list, we show results for a similar machine with the same processor (third column in the table). We note that "Sun SparcStation 10". the computer model mentioned by Hansen and Mladenović (1997) and Taillard (2003), and "Sun Ultra 30", mentioned by Senne and Lorena (2000; 2002), do not uniquely define the processor speed. In these cases, we present a range of values.

For each instance in which a method was tested, we compute the ratio between the time it required and the running time of HYBRID. Table 11.15 presents the geometric means of these ratios taken over the instances in each set (once again, only instances tested by the relevant method are considered). We believe this makes more sense than the usual arithmetic mean in this case: if a method is twice as fast as another for 50% of the instances and half as fast for the other 50%, intuitively the methods should be considered equivalent. The geometric mean reflects that, whereas the arithmetic mean does not.

One important observation regarding the values presented should be made: for VNS and VNDS, the times taken into consideration are times in which the best solution was

Table 11.14 Machines in which the various algorihms were tested.

METHOD	MACHINE USED	SIMILAR (DONGARRA, 2003)	MFLOP/S
CGLS	Sun Ultra 30	Sun UltraSparc II 250/300 MHz	114–172
DEC	Sun SparcStation 10	Sun Sparc10 or Sun Sparc10/52	10–23
HYBRID	SGI Challenge (196 MHz)	SGI Origin 2000 195 MHz	114
HYB-SS	SGI Challenge (196 MHz)	SGI Origin 2000 195 MHz	114
LOPT	Sun SparcStation 10	Sun Sparc10 or Sun Sparc10/52	10–23
LS	Sun Ultra 30	Sun UltraSparc II 250/300 MHz	114–172
VNDS	Sun Ultra I (143 MHz)	Sun Ultra 1 mod. 140	63
VNS	Sun SparcStation 10	Sun Sparc10 or Sun Sparc10/52	10–23

Table 11.15 Mean ratios between the running times obtained by methods in the literature and those obtained by HYBRID (on different machines, see Table 11.14). Smaller values are better. Values in *slanted font* indicate that there are instances in the set for which times are not available.

SERIES	HYBRID	HYB-SS	CGLS	DEC	LOPT	LSH	VNDS	VNS
GR	1.00	0.65	—	—	—	1.11	—	—
SL	1.00	0.78	0.51	—	—	24.20	—	—
ORLIB	1.00	0.90	*55.98*	—	—	*4.13*	0.46	*5.47*
fl1400	1.00	0.55	—	—	—	—	0.58	19.01
pcb3038	1.00	0.46	*9.55*	*0.21*	*0.35*	*1.67*	2.60	*30.94*
rl5934	1.00	0.48	—	—	—	—	2.93	—

found (as in the papers that describe these methods (Hansen and Mladenović, 1997; Hansen et al., 2001)); for all other algorithms (including ours), the *total* running time is considered. The values reported for our algorithm also include the time necessary to precompute all pairwise vertex distances in graph-based classes (ORLIB and SL).

Values greater than 1.00 in the table favor our method, whereas values smaller than 1.00 favor others. One cannot not take these results too literally, since they were obtained on different machines (as seen in Table 11.14). Small differences in running time should not be used to draw any conclusion regarding the relative effectiveness of the algorithms; in particular, running times within the same order of magnitude should be regarded as indistinguishable.

Based on rough estimates of the relative running times, the only methods that appear to be significantly faster than ours are DEC and LOPT, at least for the instances tested. Even though these methods (especially LOPT) can obtain solutions of reasonable quality, they are not as close to optimality as those obtained by slower methods such as ours or CGLS. Clearly, there is a trade-off between time and quality that has to be taken into account. Another particularly fast method is VNDS, which obtains solutions that are slightly worse on average than those obtained by our method, but does so in less time.

As a final note, we observe that the single-stage version of our algorithm (HYB-SS) is competitive with other methods in the graph-based classes, but lags significantly

behind for Euclidean instances (though in these cases it takes roughly half the time of the full HYBRID procedure). This shows that the post-optimization phase plays a crucial role in the robustness of HYBRID.

11.7 CONCLUDING REMARKS

This chapter presented a hybrid heuristic for the p-median problem that combines elements of several "pure" metaheuristics. It resembles GRASP in the sense that it is a multistart method in which a solution is built by a randomized constructive method and submitted to local search in each iteration. As an intensification strategy, we use path-relinking, a method originally devised for tabu search and scatter search. Solutions obtained by path-relinking, if far enough from the original extremes, are also subject to local search, which has some similarity with VNS. In the post-optimization phase, our algorithm uses the concept of multiple generations, a characteristic of genetic algorithms. We have shown that a careful combination of these elements results in a remarkably robust algorithm, capable of handling a wide variety of instances and competitive with the best heuristics in the literature.

We stress the fact that all results shown in Section 11.6 were obtained by the final version of our algorithm, with the same input parameters in all cases. The goal of the experiments shown in Sections 11.3, 11.4, and 11.5, in which various components and parameters were analyzed separately, was precisely to identify parameters that are robust enough to handle different kinds of instances, with no need for extra class-specific tuning. The tests were presented as a means to justify the decisions we made, and are not meant to be repeated by the end user. Although some gains could be obtained by additional tuning, we believe they would be very minor, and not worth the effort. The only two parameters whose change would significantly alter the behavior of the algorithm are the number of iterations and of elite solutions (these parameters were set to 32 and 10, respectively, in Section 11.6). The effect in both cases is predictable: an increase in any of these parameters would very likely result in better solutions at the expense of higher running times. Given these considerations, we believe our heuristic is a valuable candidate to be a general-purpose solver for the p-median problem. As such, the program is available from the authors upon request, or it can be directly downloaded from `http://www.research.att.com/~mgcr/popstar/`.

We do not claim, of course, that our method is the best in every circumstance. Other methods described in the literature can produce results of remarkably good quality, often at the expense of somewhat higher running times. VNS (Hansen and Mladenović, 1997) is especially successful for graph instances; VNDS (Hansen et al., 2001) is particularly strong for Euclidean instances, and is often significantly faster than our method (especially when the number of facilities to open is very small); and CGLS (Senne and Lorena, 2002), which can obtain very good results for Euclidean instances, has the additional advantage of providing good lower bounds. LOPT (Taillard, 2003) is significantly faster than our method for TSP instances, while still obtaining reasonably good solutions. After the preliminary version of our paper appeared (Resende and Werneck, 2002), at least two algorithms worthy of notice have been published. García-López et al. (2003) suggest a parallel scatter search heuristic that obtains excellent results on instance fl1400 (even improving some of the upper bounds

shown in Table 11.6), but with much higher running times. Avella et al. (2003) developed a branch-and-cut-and-price algorithm for the *p*-median problem that can solve large instances to optimality. Failing to do that, at the very least it can provide very good approximations. This method is very competitive in terms of both solution quality and runnning times. The reader is referred to their paper for a direct comparison with HYBRID.

The goal of our method is to produce close-to-optimal solutions. Therefore, it should be said that it does not handle well really large instances. If the input is a graph with millions of vertices, simply computing all-pairs shortest paths would be prohibitively slow. For that purpose, one would probably be better off relying on methods based on sampling techniques like the one proposed by Thorup (2001). Their aim is to find solutions that are "good", not near-optimal, in a reasonable (quasi-linear) amount of time. However, if one is interested in solving instances large enough to preclude the application of exact algorithms, but not so large so as to make anything worse than quasi-linear prohibitive, our method has proven to be a very worthy alternative.

An interesting research topic would be to combine elements in this paper with those of alternative heuristics for the *p*-median problem. For example, the fast implementation of the local search procedure could be used within VNS, LSH, or CGLS. The combination of elite solutions through path-relinking could be used with any method that generates a population of solutions, such as VNS, VNDS, or tabu search. LOPT and DEC, which are significantly faster than our method, could be used instead of the randomized constructive algorithm in the multistart phase of our algorithm.

Some of the ideas proposed here may even have applications beyond the *p*-median and related location problems. In particular, we believe the modifications we proposed to standard path-relinking are worthy of deeper investigation. Our algorithm benefited from strategies that improve diversity: selecting solutions from the pool in a biased way, returning a local minimum in the path if no improving solution is found, and applying local search to the solution returned. These strategies, combined with multi-generation path-relinking, can be easily incorporated into traditional metaheuristics with wide application, such as GRASP, tabu search, and VNS.

Bibliography

R. M. Aiex, M. G. C. Resende, P. M. Pardalos, and G. Toraldo. GRASP with path relinking for the three-index assignment problem. *INFORMS Journal on Computing*, 2003. To appear.

P. Avella, A. Sassano, and I. Vasil'ev. Computational study of large-scale *p*-median problems. Technical Report 08-03, DIS — Università di Roma "La Sapienza", 2003.

J. E. Beasley. A note on solving large *p*-median problems. *European Journal of Operational Research*, 21:270–273, 1985.

G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: An analytical study of exact and approximate algorithms. *Management Science*, 23:789–810, 1977.

J. J. Dongarra. Performance of various computers using standard linear equations software. Technical Report CS-89-85, Computer Science Department, University of Tennessee, 2003.

O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194:229–237, 1999.

T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

R. D. Galvão. A dual-bounded algorithm for the *p*-median problem. *Operations Research*, 28:1112–1121, 1980.

R. D. Galvão and C. S. ReVelle. A Lagrangean heuristic for the maximal covering problem. *European Journal of Operational Research*, 18:114–123, 1996.

F. García-López, B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega. The parallel variable neighborhood search for the *p*-median problem. *Journal of Heuristics*, 8(3):375–388, 2002.

F. García-López, B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega. Parallelization of the scatter search for the *p*-median problem. *Parallel Computing*, 29(5):575–589, 2003.

F. Glover. Tabu search and adaptive memory programming: Advances, applications and challenges. In R. S. Barr, R. V. Helgason, and J. L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

D. E. Goldberg. *Genetic Algorithmis in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

M. F. Goodchild and V. Noronha. Location-allocation for small computers. Monograph 8, Department of Geography, University of Iowa, 1983.

P. Hansen and N. Mladenović. Variable neighborhood search for the *p*-median. *Location Science*, 5:207–226, 1997.

P. Hansen, N. Mladenović, and D. Perez-Brito. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(3):335–350, 2001.

M. J. Hodgson. Toward more realistic allocation in location-allocation models: An interaction approach. *Environment and Planning A*, 10:1273–85, 1978.

O. Kariv and L. Hakimi. An algorithmic approach to nework location problems, part ii: The *p*-medians. *SIAM Journal of Applied Mathematics*, 37(3):539–560, 1979.

A. A. Kuehn and M. J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9(4):643–666, 1963.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

F. E. Maranzana. On the location of supply points to minimize transportation costs. *Operations Research Quarterly*, 15(3):261–270, 1964.

M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, second edition, 1994.

N. Mladenović and P. Hansen. Variable neighbourhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

R. M. Nauss and R. E. Markland. Theory and application of an optimizing procedure for lock box location analysis. *Management Science*, 27:855–865, 1981.

M. R. Rao. Cluster analysis and mathematical programming. *Journal of the American Statistical Association*, 66(335):622–626, 1971.

G. Reinelt. TSPLIB: A traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991. *http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.*

M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer, 2003.

M. G. C. Resende and R. F. Werneck. A GRASP with path-relinking for the *p*-median problem. Technical Report TD-5E53XL, AT&T Labs Research, 2002.

M. G. C. Resende and R. F. Werneck. On the implementation of a swap-based local search procedure for the *p*-median problem. In R. E. Ladner, editor, *Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments (ALENEX'03)*, pages 119–127. SIAM, 2003.

C. C. Ribeiro, E. Uchoa, and R. F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14(3):228–246, 2002.

E. Rolland, D. A. Schilling, and J. R. Current. An efficient tabu search procedure for the *p*-median problem. *European Journal of Operational Research*, 96:329–342, 1996.

K. E. Rosing. An empirical investigation of the effectiveness of a vertex substitution heuristic. *Environment and Planning B*, 24:59–67, 1997.

K. E. Rosing and C. S. ReVelle. Heuristic concentration: Two stage solution construction. *European Journal of Operational Research*, 97:75–86, 1997.

K. E. Rosing, C. S. ReVelle, E. Rolland, D. A. Schilling, and J. R. Current. Heuristic concentration and tabu search: A head to head comparison. *European Journal of Operational Research*, 104:93–99, 1998.

K. E. Rosing, C. S. ReVelle, and H. Rosing-Vogelaar. The *p*-median and its linear programming relaxation: An approach to large problems. *Journal of the Operational Research Society*, 30(9):815–823, 1979.

E. L. F. Senne, 2002. Personal communication.

E. L. F. Senne and L. A. N. Lorena. Langrangean/surrogate heuristics for *p*-median problems. In M. Laguna and J. L. González-Velarde, editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pages 115–130. Kluwer, 2000.

E. L. F. Senne and L. A. N. Lorena. Stabilizing column generation using Lagrangean/surrogate relaxation: an application to *p*-median location problems. *European Journal of Operational Research*, 2002. To appear.

E. D. Taillard. Heuristic methods for large centroid clustering problems. *Journal of Heuristics*, 9(1):51–74, 2003.

B. C. Tansel, R. L. Francis, and T. J. Lowe. Location on networks: a survey. *Management Science*, 29(4):482–511, 1983.

M. B. Teitz and P. Bart. Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, 16(5):955–961, 1968.

M. Thorup. Quick *k*-median, *k*-center, and facility location for sparse graphs. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP 2001)*, volume 2076 of *Lecture Notes in Computer Science*, pages 249–260. Springer, 2001.

H. D. Vinod. Integer programming and the theory of groups. *Journal of the American Statistical Association*, 64(326):506–519, 1969.

S. Voß. A reverse elimination approach for the *p*-median problem. *Studies in Locational Analysis*, 8:49–58, 1996.

R. Whitaker. A fast algorithm for the greedy interchange of large-scale clustering and median location prolems. *INFOR*, 21:95–108, 1983.

12 A HYBRID MULTISTART HEURISTIC FOR THE UNCAPACITATED FACILITY LOCATION PROBLEM

Mauricio G. C. Resende[1] and Renato F. Werneck[2]

[1]Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA.
mgcr@research.att.com

[2]Department of Computer Science
Princeton University
Princeton, NJ 08544 USA
rwerneck@cs.princeton.edu

Abstract: We present a multistart heuristic for the uncapacitated facility location problem, based on a very successful method we originally developed for the p-median problem. We show extensive empirical evidence to the effectiveness of our algorithm in practice. For most benchmarks instances in the literature, we obtain solutions that are either optimal or a fraction of a percentage point away from it. Even for pathological instances (created with the sole purpose of being hard to tackle), our algorithm can get very close to optimality if given enough time. It consistently outperforms other heuristics in the literature.
Keywords: GRASP, location theory, facility location.

12.1 INTRODUCTION

Consider a set F of *potential facilities*, each with a *setup cost* $c(f)$, and let U be a set of *users* (or *customers*) that must be served by these facilities. The cost of serving user u with facility f is given by the *distance* $d(u, f)$ between them (often referred to as *service cost* or *connection cost* as well). The *facility location problem* consists in determining a set $S \subseteq F$ of facilities to open so as to minimize the total cost (including

setup and service) of covering all customers:

$$\text{cost}(S) = \sum_{f \in S} c(f) + \sum_{u \in U} \min_{f \in S} d(u, f).$$

Note that we assume that each user is allocated to the closest open facility, and that this is the *uncapacitated* version of the problem: there is no limit to the number of users a facility can serve. Even with this assumption, the problem is NP-hard (Cornuéjols et al., 1990).

This is perhaps the most common location problem, having been widely studied in the literature, both in theory and in practice.

Exact algorithms for this problem do exist (some examples are Conn and Cornueéjols (1990); Körkel (1989)), but the NP-hard nature of the problem makes heuristics the natural choice for larger instances.

Ideally, one would like to find heuristics with good performance guarantees. Indeed, much progress has been made in terms of approximation algorithms for the metric version of this problem (in which all distances obey the triangle inequality). Shmoys et al. (1997) presented the first polynomial-time algorithm with a constant approximation factor (approximately 3.16). Several improved algorithms have been developed since then, with some of the latest (Jain et al., 2003; 2002; Mahdian et al., 2002) being able to find solutions within a factor of around 1.5 from the optimum. Unfortunately, there is not much room for improvement in this area. Guha and Khuller (1999) have established a lower bound of 1.463 for the approximation factor, under some widely believed assumptions.

In practice, however, these algorithms tend to be much closer to optimality for non-pathological instances. The best algorithm proposed by Jain et al. (2003), for example, has a performance guarantee of only 1.61, but was always within 2% of optimality in their experimental evaluation.

Although interesting in theory, approximation algorithms are often outperformed in practice by more straightforward heuristics with no particular performance guarantees. Constructive algorithms and local search methods for this problem have been used for decades, since the pioneering work of Kuehn and Hamburger (1963). Since then, more sophisticated metaheuristics have been applied, such as simulated annealing (Alves and Almeida, 1992), genetic algorithms (Kratica et al., 2001), tabu search (Ghosh, 2003b; Michel and Van Hentenryck, 2003), and the so-called "complete local search with memory" (Ghosh, 2003b). Dual-based methods, such as Erlenkotter's dual ascent (Erlenkotter, 1978), Guignard's Lagragean dual ascent (Guignard, 1988), and Barahona and Chudak's volume algorithm (Barahona and Chudak, 1999) have also shown promising results.

An experimental comparison of some state-of-the-art heuristics is presented by Hoefer (2003) (slightly more detailed results are presented in Hoefer (2002a)). Five algorithms are tested: JMS, an approximation algorithm presented by Jain et al. (2002); MYZ, also an approximation algorithm, this one by Mahdian et al. (2002); swap-based local search; Michel and Van Hentenryck's tabu search (Michel and Van Hentenryck, 2003); and the volume algorithm (Barahona and Chudak, 1999). Hoefer's conclusion, based on experimental evidence, is that tabu search finds the best solutions within reasonable time, and recommends this method for practitioners.

```
function HYBRID (seed, maxit, elitesize)
1     randomize(seed);
2     init(elite, elitesize);
3     for i = 1 to maxit do
4           S ← randomizedBuild();
5           S ← localSearch(S);
6           S′ ← select(elite, S);
7           if (S′ ≠ NULL) then
8                 S′ ← pathRelinking(S, S′);
9                 add(elite, S′);
10          endif
11          add(elite, S);
12    endfor
13    S ← postOptimize(elite);
14    return S;
end HYBRID
```

Figure 12.1 Pseudocode for HYBRID, as given in Resende and Werneck (2004).

In this paper, we provide an alternative that can be even better in practice. It is a hybrid multistart heuristic akin to the one we developed for the *p*-median problem in Resende and Werneck (2004). A series of minor adaptations is enough to build a very robust algorithm for the facility location problem, capable of obtaining near-optimal solutions for a wide variety of instances of the facility location problem.

The remainder of the paper is organized as follows. In Section 12.2, we describe our algorithm and its constituent parts. Section 12.3 presents empirical evidence to the effectiveness of our method, including a comparison with Michel and Van Hentenryck's tabu search. Final remarks are made in Section 12.4.

12.2 THE ALGORITHM

In Resende and Werneck (2004), we introduce a new hybrid metaheuristic and apply it to the *p*-median problem. Figure 12.1 reproduces the outline of the algorithm, exactly as presented there.

The method works in two phases. The first is a multistart routine with intensification. In each iteration, it builds a randomized solution and applies local search to it. The resulting solution (S) is combined, through a process called *path-relinking*, with some other solution from a pool of *elite solutions* (which represents the best solutions found thus far). This results in a new solution S'. The algorithm then tries to insert both S' and S into the pool; whether any of those is actually inserted depends on its value, among other factors. The second is a post-optimization phase, in which the solutions in the pool of elite solutions are combined among themselves in a process that hopefully results in even better solutions.

We call this method HYBRID because it combines elements of several other metaheuristics, such as scatter and tabu search (which make heavy use of path-relinking)

and genetic algorithms (from which we take the notion of generations). A more detailed analysis of these similarities is presented in Resende and Werneck (2004).

Of course, Figure 12.1 presents only the outline of an algorithm. Many details are left to be specified, including which problem it is supposed to solve. Although originally proposed for the p-median problem, there is no specific mention to it in the code, and in fact the same framework could be applied to other problems. In this paper, our choice is facility location.

Recall that the p-median problem is very similar to facility location: the only difference is that, instead of assigning costs to facilities, the p-median problem must specify p, the exact number of facilities that must be opened. With minor adaptations, we can reuse several of the components used in Resende and Werneck (2004), such as the constructive algorithm, local search, and path-relinking.

The adaptation of the p-median heuristic shown in this paper is as straightforward as possible. Although some problem-specific tuning could lead to better results, the potential difference is unlikely to be worth the effort. We therefore settle for simple, easy-to-code variations of the original method.

Constructive heuristic.. In each iteration i, we first define the number of facilities p_i that will be open. This number is $\lceil m/2 \rceil$ in the first iteration; for $i > 1$, we pick the average number of facilities in the solutions found (after local search) in the first $i-1$ iterations. Now that we have p_i, we execute the sample procedure exactly as described in Resende and Werneck (2004). It adds facilities one by one. In each step, the algorithm chooses $\lceil \log_2(m/p_i) \rceil$ facilities uniformly at random and selects the one among those that reduces the total service cost the most.

Local search.. The local search used in Resende and Werneck (2004) is based on swapping facilities. Given a solution S, we look for two facilities, $f_r \in S$ and $f_i \notin S$, which, if swapped, lead to a better solution. A property of this method is that it keeps the number of open facilities constant. This is required for the p-median problem, but not for facility location, so in this paper we also allow "pure" insertions and deletions (in addition to swaps). All possible insertions, deletions, and swaps are considered, and the best among those is performed. The local search stops when no improving move exists, in which case the current solution is a *local minimum* (or *local optimum*). This local search is known as *flip* + *swap* (Kochetov, 2003).

The actual implementation of the local search is essentially the same used in Resende and Werneck (2004) (and described in detail in Resende and Werneck (2003a;b)) for the p-median problem. We briefly recall the main ideas here. Let $profit(f_i, f_r)$ be the amount by which the solution value is reduced if f_i is the facility inserted and f_r the one removed. The algorithm computes the profit associated to every pair (f_i, f_r) (for all $f_i \in S$ and all $f_r \notin S$). If the maximum profit is positive, we perform the corresponding swap and repeat; otherwise, we stop. The computation is divided into three components:

- $save(f_i)$: decrease in solution value due to the insertion of f_i (with no associated removal);

- $loss(f_r)$: increase in solution value due to the removal of f_r (with no associated insertion);

- $extra(f_i, f_r)$: a positive correction term that accounts for the fact that the insertion of f_i and the removal of f_r may not be independent (a user previously assigned to f_r may be reassigned to f_i); the definition of *extra* is such that the following relation holds:

$$profit(f_i, f_r) = save(f_i) - loss(f_r) + extra(f_i, f_r).$$

Instead of computing these values from scratch in each iteration, our implementation just updates them from one iteration of the local search to another. To achieve this goal, *save* and *loss* are represented as arrays; *extra*, being the only term that depends on both f_i and f_r, is represented as a matrix.

It can be shown (Resende and Werneck, 2003b) that $extra(f_i, f_r)$ is nonzero only when f_i and f_r are "close" to each other.[1] One only has to worry explicitly about the nonzero terms; all others are determined implicitly. This observation is crucial for a fast implementation of the local search procedure, since it allows *extra* to be represented as a sparse matrix. In instances from the literature, this implementation has been shown (Resende and Werneck, 2003b) to be up to three orders of magnitude faster than previous methods (even though its worst-case complexity, $O(mn)$, is the same) for the p-median problem.

As mentioned in Resende and Werneck (2003a), this algorithm can be adapted to the facility location problem in a very natural way. First, we must take setup costs into account, which can be accomplished simply by subtracting them from *save* and *loss*. Second, we must consider that single insertions or deletions are now valid moves (and not only swaps). But this comes for free: *save* and *loss* already represent the profits obtained with insertions and deletions, respectively. These are the only differences between the algorithms.

Path-relinking.. Path-relinking is an intensification procedure originally devised for scatter search and tabu search (Glover, 1996; Glover et al., 2000; Laguna and Martí, 1999), but often used with other methods, such as GRASP (Resende and Ribeiro, 2003; 2005). In this paper, we apply the variant described in Resende and Werneck (2004). It takes two solutions as input, S_1 and S_2. The algorithm starts from S_1 and gradually transforms it into S_2. The operations that change the solution in each step are the same used in the local search: insertions, deletions, and swaps. In this case, however, only facilities in $S_2 \setminus S_1$ can be inserted, and only those in $S_1 \setminus S_2$ can be removed. In each step, the most profitable (or least costly) move—considering all three kinds—is performed. The procedure returns the best local optimum in the path from S_1 to S_2. If no local optimum exists, one of the extremes is chosen with equal probability.

[1] More precisely, when there is at least one user for which f_i is closest than the second closest facility in the original solution.

Elite solutions.. The `add` operation in Figure 12.1 must decide whether a new solution should be inserted into the pool or not. The criteria we use here are similar to those proposed in Resende and Werneck (2004). They are based on the notion of *symmetric difference* between two solutions S_a and S_b, defined as $|S_a \setminus S_b| + |S_b \setminus S_a|$.[2] A new solution will be inserted into the pool only if its symmetric difference to each cheaper solution already there is at least four. Moreover, if the pool is full, the new solution must also cost less than the most expensive element in the pool; in that case, the new solution replaces the one (among those of equal or greater cost) it is most similar to.

Intensification.. After each iteration, the solution S obtained by the local search procedure is combined (with path-relinking) with a solution S' obtained from the pool, as shown in line 8 of Figure 12.1. Solution S' is chosen at random, with probability proportional to its symmetric difference to S. Path-relinking is always performed from S to S'.

Post-optimization.. Once the multistart phase is over, all elite solutions are combined with one another, also with path-relinking (within each pair, path-relinking is performed from the best to the worst solution). The solutions thus produced are used to create a new pool of elite solutions (subject to the same rules as in the original pool), to which we refer as a new *generation*. If the best solution in the new generation is strictly better than the best previously found, we repeat the procedure. This process continues until a generation that does not improve upon the previous one is created. The best solution found across all generations is returned as the final result of the algorithm.

Parameters.. As the outline in Figure 12.1 shows, the procedure takes only two input parameters (other than the random seed): the number of iterations in the multistart phase and the size of the pool of elite solutions. In Resende and Werneck (2004), we set those values to 32 and 10, respectively. In the spirit of keeping changes to a minimum, we use the same values here for the "standard version" of our algorithm.

Whenever we need versions of our algorithm with shorter or longer running times (to ensure a fair comparison with other methods), we change both parameters. Recall that the running time of the multistart phase of the algorithm depends linearly on the number of iterations, whereas the post-optimization phase depends quadratically (roughly) on the number of elite solutions (because all solutions are combined among themselves). Therefore, if we want to multiply the average running time of the algorithm by some factor x, we just multiply the number of multistart iterations by x and the number of elite solutions by $\sqrt{x}$ (rounding appropriately).

We observe that running time and solution quality are determined by several design choices, and not only the number of iterations and of elite solutions. Consider the intensification strategies, for instance. To reduce the running time of the algorithm, we

[2]This definition is slightly different from the one we used for the p-median problem, since now different solutions need not have the same number of facilities.

could decide not to run the post-optimization phase, or not to run path-relinking during the multistart phase (or not at all). Or, to increase solution quality, we could consider performing path-relinking between two solutions S_1 and S_2 in both ways (from S_1 to S_2 and from S_2 to S_1) and picking the best. We could also try other constructive heuristics. These and other variants of the algorithm are studied in the context of the p-median problem in Resende and Werneck (2004). The variant reported here achieved the best balance overall between running time and solution quality. The most important aspects of the algorithm are the fast implementation of the local search and the use of path-relinking. Other aspects, such as the constructive heuristic, the methods for maintaining the pool of elite solutions, and the precise criteria for adding and removing solutions from it played relatively minor roles.

12.3 EMPIRICAL RESULTS

12.3.1 Experimental Setup

The algorithm was implemented in `C++` and compiled with the SGI MIPSPro C++ compiler (v. 7.30) with flags `-O3 -OPT:Olimit=6586`. The program was run on an SGI Challenge with 28 196-MHz MIPS R10000 processors, but each execution was limited to a single processor. All times reported are CPU times measured by the `getrusage` function with a precision of 1/60 second. The random number generator we used was Matsumoto and Nishimura's *Mersenne Twister* (Matsumoto and Nishimura, 1998). The source code for the algorithm is available from the authors upon request.

The algorithm was tested on all classes from the UflLib (Hoefer, 2003) at the time of writing and on class GHOSH, described in Ghosh (2003b). In every case, the number of users is the same as the number of potential facilities. The reader is referred to Hoefer (2002b) and Ghosh (2003b) for detailed descriptions of each class. A brief overview is presented below:

- BK: Generated based on the description provided by Bilde and Krarup (1977). There are 220 instances in total, with 30 to 100 users. Connection costs are always picked uniformly at random from $[0, 1000]$. Setup costs are always at least 1000, but the exact range depends on the subclass (there are 22 of those, with 10 instances each).

- FPP: Class introduced by Kochetov (2003); Kochetov and Ivanenko (2003). Each instance corresponds to a finite projective plane of order k, with $n = k^2 + k + 1$ points and n lines. In the UFL instance, the distance between i and j is an integer between 0 and 4 if point j is on line i, and infinity otherwise; at most $n+1$ values are finite. Setup costs are 3000. There are two subclasses, each with 40 instances: FPP11 (with $k = 11$ and $n = 133$) and FPP17 (with $k = 17$ and $n = 307$). Although optimal solutions in this class can be found in polynomial time, the instances are hard for algorithms based on the flip+swap local search, since each instance has a large number of strong local optima and the distance between them is at least $2k$ (Kochetov, 2003).

- GAP: Also designed by Kochetov (2003); Kochetov and Ivanenko (2003), these instances have large duality gaps, usually greater than 20%. They are hard especially for dual-based methods. Setup costs are always 3000. The service cost associated with each facility is infinity for most customers, and between 0 and 5 for the remaining few (the end result resembles a set covering instance). There are three subclasses (GAPA, GAPB, and GAPC), each with 30 instances. Each customer in GAPA is covered by 10 facilities (i.e., the service cost for all others is infinity); each facility in GAPB covers exactly 10 customers; subclass GAPC (the hardest) combines both constraints: each customer is covered by 10 facilities, and each facility covers 10 customers. On all cases, assignments are made at random.

- GHOSH: Class created by Ghosh (2003b), following the guidelines set up by in Körkel (1989). There are 90 instances in total, with $n = m$ on all cases. They are divided into two groups of 45 instances, one symmetric and the other asymmetric. Each group contains three values of n: 250, 500, and 750.[3] Connection costs are integers taken uniformly at random from $[1000, 2000]$. For each value of n there are three subclasses, each with five instances; they differ in the range of values from which setup costs are drawn: it can be $[100, 200]$ (range A), $[1000, 2000]$ (B) or $[10000, 20000]$ (C). Each subclass is named after its parameters: GS250B, for example, is symmetric, has 250 nodes, and service costs ranging from 1000 to 2000.

- GR: Graph-based instances by Galvão and Raggi (1989). The number of users is either 50, 70, 100, 150, or 200. There are 50 instances in total, 10 for each value of n. Connection costs are given by the corresponding shortest paths in the underlying graph. (Instances are actually given as distance matrices, so there is no overhead associated with computing shortest paths.)

- M*: This class was created with the generator introduced by Kratica et al. (2001). These instances have several near-optimal solutions, which according the authors makes them close to "real-life" applications. There are 22 instances in this class, with n ranging from 100 to 2000.

- MED: Originally proposed for the p-median problem by Ahn et al. (1998), these instances were later used in the context of uncapacitated facility location by Barahona and Chudak (1999). Each instance is a set of n points picked uniformly at random in the unit square. A point represents both a user and a potential facility, and connection costs are determined by the corresponding Euclidean distances. All values are rounded up to 4 significant digits and made integer (Hoefer, 2003). Six values of n were used: 500, 1000, 1500, 2000, 2500, and 3000. In each case, three different setup costs were tested: $\sqrt{n}/10$, $\sqrt{n}/100$, and $\sqrt{n}/1000$.

[3] These are actually the three largest values tested in Ghosh (2003b); some smaller instances are tested there as well.

- ORLIB: These instances are part of Beasley's OR-Library (Beasley, 1990). Originally proposed as instances for the capacitated version of the facility location problem in Beasley (1993), they can be used in the uncapacitated setting as well (one just has to ignore the capacities).

All instances were downloaded from the UflLib website (Hoefer, 2002b), with the exception of those in class GHOSH, created with a generator kindly provided by Ghosh (2003a). Five of these eight classes were used in Hoefer's comparative analysis (Hoefer, 2002a; 2003): BK, GR, M*, MED, and ORLIB.

12.3.2 Results

12.3.2.1 Quality Assessment. As already mentioned, the "standard" version of our algorithm has 32 multistart iterations and 10 elite solutions. It was run ten times on each instance available, with ten different random seeds (1 to 10).

Although more complete data will be presented later in this section, we start with a broad overview of the results we obtained. Table 12.1 shows the average deviation (in percentage terms) obtained by our algorithm with respect to the best known bounds. All optima are known for FPP, GAP, BK, GR, and ORLIB. We used the best upper bounds shown in Hoefer (2002b) for MED and M* (upper bounds that are not proved optimal were obtained either by tabu search or local search). For GHOSH, we used the bounds shown in Ghosh (2003b); some were obtained by tabu search, others by complete local search with memory. Table 12.1 also shows the mean running times obtained by our algorithm. To avoid giving too much weight to larger instances, we used geometric means for times.

Table 12.1 Average deviation with respect to the best known upper bounds and mean running times of HYBRID (with 32 iterations and 10 elite solutions) for each class.

CLASS	AVG%DEV	TIME (S)
BK	0.002	0.28
FPP	33.375	7.66
GAP	5.953	1.64
GHOSH	-0.039	34.31
GR	0.000	0.32
M*	0.000	7.86
MED	-0.391	369.67
ORLIB	0.000	0.17

In terms of solution quality, our algorithm does exceedingly well for all five classes tested in Hoefer (2002a). It matched the best known bounds (usually the optimum) on every single run of GR, M*, and ORLIB. The algorithm did have a few unlucky runs on class BK, but the average error was still only 0.001%. On MED, the solutions it found were on average 0.4% better than the best upper bounds shown in Hoefer (2002a).

Our method also handles very well the only class not in the UflLib, GHOSH. It found solutions at least as good as the best in Ghosh (2003b). This is especially relevant considering that we are actually comparing our results with the best among *two* algorithms in each case (tabu search and complete local search with memory).

The two remaining classes, GAP and FPP, were created with the intent of being hard. At least for our algorithm, they definitely are: on average, solutions were within 28% and 6% from optimality, respectively. This is several orders of magnitude worse than the results obtained for other classes. However, as Subsection 12.3.2.2 will show, the algorithm can obtain solutions of much better quality if given more time.

Detailed results.. For completeness, Tables 12.2 to 12.8 show the detailed results obtained HYBRID on each of the eight classes of instances. They refer to the exact same runs used to create Table 12.1.

Tables 12.2 and 12.3 show the results for M* and ORLIB, respectively. For each instance, we show the best known bounds (which were matched by our algorithm on all runs on both classes) and the average running time.

Table 12.2 Results for M* instances. Average solution values for HYBRID and mean running times (with 32 iterations and 10 elite solutions). All runs matched the best bounds shown in Hoefer (2002a).

NAME	n	VALUE	TIME (S)
mo1	100	1305.95	0.988
mo2	100	1432.36	1.030
mo3	100	1516.77	0.960
mo4	100	1442.24	0.892
mo5	100	1408.77	0.815
mp1	200	2686.48	3.695
mp2	200	2904.86	4.125
mp3	200	2623.71	3.500
mp4	200	2938.75	3.887
mp5	200	2932.33	4.169
mq1	300	4091.01	8.919
mq2	300	4028.33	7.802
mq3	300	4275.43	9.508
mq4	300	4235.15	9.834
mq5	300	4080.74	10.813
mr1	500	2608.15	27.221
mr2	500	2654.74	27.646
mr3	500	2788.25	26.417
mr4	500	2756.04	27.595
mr5	500	2505.05	26.989
ms1	1000	5283.76	113.395
mt1	2000	10069.80	701.167

Table 12.3 Results for ORLIB instances. Average running times for HYBRID with 32 iterations and 10 elite solutions. The optimum solution value was found on all runs.

NAME	n	OPTIMUM	TIME (S)
cap101	50	796648.44	0.055
cap102	50	854704.20	0.056
cap103	50	893782.11	0.072
cap104	50	928941.75	0.077
cap131	50	793439.56	0.105
cap132	50	851495.32	0.097
cap133	50	893076.71	0.131
cap134	50	928941.75	0.140
cap71	50	932615.75	0.034
cap72	50	977799.40	0.039
cap73	50	1010641.45	0.053
cap74	50	1034976.97	0.049
capa	1000	17156454.48	7.380
capb	1000	12979071.58	6.245
capc	1000	11505594.33	6.148

Results for class MED are shown in Table 12.4. For each instance, we present the best known lower and upper bounds, as given in Table 12 of Hoefer (2002a). Lower bounds were found by the volume algorithm (Barahona and Chudak, 1999), and upper bounds by either local search or tabu search (Michel and Van Hentenryck, 2003), depending on the instance. The average solution value obtained by HYBRID in each case is shown in Table 12.4 in absolute and percentage terms (in the latter case, when compared with both lower and upper bounds). On average, HYBRID found solutions that are at least 0.15% better than previous bounds, and sometimes the gains were upwards of 0.5%. In fact, our results were in all cases much closer to the lower bound than to previous upper bounds. Average solution values are within 0.178% or less from optimality, possibly better (depending on how good the lower bounds are).

Since classes BK and GR have more instances (220 and 50, respectively), we aggregate them into subclasses. Each subclass contains 10 instances built with the exact same parameters (such as number of elements and cost distribution), just with different random seeds. Table 12.5 presents the results for BK: for each subclass, we present the average error obtained by the algorithm and the average running time. Table 12.6 refers to class GR and presents the average running times only, since the optimal solution was found in every single run.

Table 12.7 shows the results for class GHOSH, which is divided into 5-instance subclasses. The table shows the best bounds found in Ghosh (2003b), by either tabu search or complete local search with memory (we picked the best in each case). For reference, we also show the running times reported in Ghosh (2003b), but the reader

Table 12.4 Results for MED instances. Columns 2 and 3 show the best known lower and upper bounds, as given in Tables 11 and 12 of Hoefer (2002a). The next three columns show the quality obtained by HYBRID: first the average solution value, then the average percentage deviation from the lower and upper bounds, respectively. The last column shows the average running times of our method.

NAME	LOWER	UPPER	AVERAGE	AVG%L	AVG%U	TIME (S)
med0500-10	798399	800479	798577.0	0.022	-0.238	33.2
med0500-100	326754	328540	326805.4	0.016	-0.528	32.9
med0500-1000	99099	99325	99169.0	0.071	-0.157	23.6
med1000-10	1432737	1439285	1434185.4	0.101	-0.354	173.9
med1000-100	607591	609578	607880.4	0.048	-0.278	148.8
med1000-1000	220479	221736	220560.9	0.037	-0.530	141.7
med1500-10	1997302	2005877	2001121.7	0.191	-0.237	347.8
med1500-100	866231	870182	866493.2	0.030	-0.424	378.7
med1500-1000	334859	336263	334973.2	0.034	-0.384	387.2
med2000-10	2556794	2570231	2558120.8	0.052	-0.471	717.5
med2000-100	1122455	1128392	1122861.9	0.036	-0.490	650.8
med2000-1000	437553	439597	437690.7	0.031	-0.434	760.0
med2500-10	3095135	3114458	3100224.7	0.164	-0.457	1419.5
med2500-100	1346924	1352322	1347577.6	0.049	-0.351	1128.2
med2500-1000	534147	536546	534426.6	0.052	-0.395	1309.4
med3000-10	3567125	3586599	3570818.8	0.104	-0.440	1621.1
med3000-100	1600551	1611186	1602530.9	0.124	-0.537	1977.6
med3000-1000	643265	645680	643541.8	0.043	-0.331	2081.4

should bear in mind that they were found on a machine with a different processor (an Intel Mobile Celeron running at 650 MHz).[4]

The last three columns in the table report the results obtained by HYBRID: the solution value, the average deviation with respect to the upper bounds, and the running time (all three values are averages taken over the 50 runs in each subclass).

Finally, average solution qualities and running times are shown to each subclass of FPP and GAP in Table 12.8.

12.3.2.2 Comparative Analysis. We have seen that our algorithm obtains solutions of remarkable quality for most classes of instances tested. On their own, however, these results do not mean much. Any reasonably scalable algorithm should be able to find good solutions if given enough time.

[4]From Dongarra (2003), we can infer that this processor and the one we use have similar speeds, or at least within the same order of magnitude. The machine in Dongarra (2003) that is most similar to Ghosh's is a Celeron running at 433 MHz, capable of 160 Mflop/s. According to the same list, the speed of our processor is 114 Mflop/s (based on an entry for an SGI Origin 2000 at 195 MHz).

Table 12.5 Results for BK instances: average percent errors with respect to the optima and average running times of HYBRID (with 32 iterations and 10 elite solutions).

SUBCLASS	n	AVG%ERR	TIME (S)
B	100	0.0000	0.310
C	100	0.0160	0.450
D01	80	0.0001	0.223
D02	80	0.0000	0.211
D03	80	0.0000	0.199
D04	80	0.0000	0.170
D05	80	0.0000	0.162
D06	80	0.0000	0.186
D07	80	0.0000	0.174
D08	80	0.0000	0.166
D09	80	0.0000	0.175
D10	80	0.0000	0.166
E01	100	0.0000	0.476
E02	100	0.0000	0.588
E03	100	0.0188	0.512
E04	100	0.0000	0.464
E05	100	0.0000	0.376
E06	100	0.0000	0.408
E07	100	0.0000	0.416
E08	100	0.0000	0.418
E09	100	0.0000	0.352
E10	100	0.0000	0.353

With that in mind, we compare the results obtained by our algorithm with those obtained by Michel and Van Hentenryck's tabu search algorithm (Michel and Van Hentenryck, 2003), which achieved the best experimental results among the algorithms tested in (Hoefer, 2002a). We refer to this method as TABU. Starting from a random solution, in each iteration it executes a *flip* operation, i.e., it opens or closes an individual facility. This defines a neighborhood that is more restricted than the one we use, which also allows swaps. While the best neighbor can be found considerably faster, the local search tends to reach local optima sooner. To escape them, the method uses a tabu list, which forbids facilities recently inserted or removed from being flipped. The algorithm stops after executing 500 consecutive iterations without an improvement in the objective function

We downloaded the source code for an implementation of TABU from the UflLib. To ensure that running times are comparable, we compiled it with the same parameters used for HYBRID and ran the program on the same machine. Since TABU has a randomized component (the initial solution), we ran it 10 times for each instance in the class, with different seeds for the random number generator (the seeds were 1 to 10).

Table 12.6 Results for GR instances: average running times of HYBRID (with 32 iterations and 10 elite solutions) as a function of n (each subclass contains 10 instances). Every execution found the optimal solution.

n	TIME (S)
50	0.098
70	0.163
100	0.308
150	0.602
200	1.123

As suggested in Michel and Van Hentenryck (2003), the algorithm was run with 500 non-improving consecutive iterations as the stopping criterion. However, with this number of iterations TABU is much faster than the standard version of HYBRID (with 32 iterations and 10 elite solutions). For a fair comparison, we also ran a faster version of our method, with only 8 iterations and 5 elite solutions. The results obtained by this variant of HYBRID and by TABU are summarized in Table 12.9. For each class, the average solution quality (as the percentage deviation with respect to the upper bound in Hoefer (2002a)) and the mean running times are shown.

Note that both algorithms have similar running times, much lower than those presented in Table 12.1. Even so, both algorithms find solutions very close to the optimal (or best known) on five classes: BK, GHOSH, GR, M*, and ORLIB. Although in all cases HYBRID found slightly better solutions, both methods performed rather well: on average, TABU was always within 0.1% of the best previously known bounds, and HYBRID was within 0.03%. Our algorithm was actually able to improve the bounds for GHOSH (presented in Ghosh (2003b)), whereas TABU could only match them (albeit in slightly less time).

Although there are some minor differences between the algorithms for these five classes, it is not clear which is best. Both usually find the optimal values in comparable times. In a sense, these instances are just too easy for either method. We need to look at the remaining classes to draw any meaningful conclusion.

Consider class MED. Both algorithms run for essentially the same time on average. TABU almost matches the best bounds presented in Hoefer (2002a). This was expected, since most of those bounds were obtained by TABU itself (a few were established by local search). However, HYBRID does much better: on average, the solutions it finds are 0.364% below the reference upper bounds.

Even greater differences were observed for GAP and FPP: these instances are meant to be hard, and indeed they are for both algorithms. On average, the solutions HYBRID found for GAP instances were almost 10% off optimality; TABU did even worse, with an average error of 16%. The hardest class is FPP: the average deviation from optimality was almost 70% for our algorithm, and more than 95% for TABU. Even though HYBRID does slightly better than TABU on both classes, the results it provides are hardly satisfactory for a method that is supposed to find near-optimal solutions.

Table 12.7 Results for GHOSH instances. The upper bounds are the best reported by (Ghosh, 2003b), with the corresponding running times (obtained on a different machine, an Intel Mobile Celeron running at 650 MHz). The results for HYBRID (with 32 iterations and 10 elite solutions) are shown in the last three columns.

INSTANCE		UPPER BOUND (GHOSH, 2003B)		HYBRID		
NAME	n	VALUE	TIME (S)	VALUE	AVG%DEV	TIME (S)
GA250A	250	257978.4	18.3	257922.1	-0.022	5.7
GA250B	250	276184.2	6.5	276053.6	-0.047	8.2
GA250C	250	333058.4	17.3	332897.2	-0.048	7.4
GA500A	500	511251.6	18.1	511147.4	-0.020	40.3
GA500B	500	538144.0	6.4	537868.2	-0.051	52.2
GA500C	500	621881.8	24.7	621475.2	-0.065	57.4
GA750A	750	763840.4	213.3	763741.0	-0.013	117.5
GA750B	750	796754.2	71.4	796393.5	-0.045	127.1
GA750C	750	900349.8	146.5	900198.6	-0.017	136.5
GS250A	250	257832.6	207.1	257807.9	-0.010	5.3
GS250B	250	276185.2	79.2	276035.2	-0.054	8.0
GS250C	250	333671.6	134.6	333671.6	0.000	8.3
GS500A	500	511383.6	824.3	511203.0	-0.035	43.5
GS500B	500	538480.4	409.4	537919.1	-0.104	52.6
GS500C	500	621107.2	347.4	621059.2	-0.008	50.8
GS750A	750	763831.2	843.2	763713.9	-0.015	112.6
GS750B	750	796919.0	396.0	796593.7	-0.041	126.3
GS750C	750	901158.4	499.7	900183.8	-0.108	130.3

Table 12.8 Results for FPP and GAP instances: average percent errors (with respect to the optimal solutions) and average running times for each subclass.

SUBCLASS	n	AVG%ERR	TIME (S)
GAPA	100	5.14	1.41
GAPB	100	5.98	1.81
GAPC	100	6.74	1.89
FPP11	133	8.48	2.58
FPP17	307	58.27	25.18

Note, however, that the mean time spent on each instance is around one second, which is not much. Being implementations of metaheuristics, both algorithms should behave much better if given more time. To test if this is indeed the case, we performed longer runs of each method.

Table 12.9 Average deviation with respect to the best known upper bounds and mean running times in each class for HYBRID (with 8 iterations and 5 elite solutions) and TABU (with 500 non-improving iterations as the stop criterion).

CLASS	HYBRID		TABU	
	AVG%DEV	TIME (S)	AVG%DEV	TIME (S)
BK	0.028	0.087	0.071	0.155
FPP	69.367	1.741	95.711	0.650
GAP	9.573	0.348	15.901	0.259
GHOSH	-0.032	8.816	0.002	4.570
GR	0.000	0.090	0.100	0.160
M*	0.004	2.196	0.011	1.750
MED	-0.364	92.387	0.073	92.854
ORLIB	0.000	0.048	0.028	0.160

We tested two different versions of tabu search. In the first variant (TABU), we vary the number of iterations in the stopping criterion: 500 (the original value), 1000, 2000, 4000, 8000, 16000, 32000, and 64000. The second variant, TABUMS, is a multistart version of the algorithm. After it reaches 500 non-improving iterations, it starts again from a new (random) solution. The best solution overall is picked as the final result. We tested this method with 1 to 128 restarts, corresponding to 500 to 64000 final non-improving iterations overall.

We also tested three different versions of our algorithm. The first is the standard version depicted in Figure 12.1, HYBRID. It has two input parameters (number of iterations and number of elite solutions), so we varied both at the same time. We tested the following pairs: 4:3, 8:5, 16:7, 32:10 (the original parameters), 64:14, 128:20, 256:28, 512:40, 1024:57, and 2048:80. Note that to move from one pair to the next we multiply the number of iterations by 2 and the number of elite solutions by $\sqrt{2}$, as mentioned in Section 12.2.

The second version is MULTISTART+PR, which is similar to HYBRID but does not execute post-optimization; however, it does execute path-relinking between multistart iterations, as in the original hybrid. The best solution found after all iterations are completed is picked as the result. We ran this algorithm with the same set of parameters (number of iterations and elite solutions) as HYBRID.

The third version we tested (MULTISTART) does not execute path-relinking at any stage: in each iteration, it finds a greedy randomized solution and executes local search on it. The best solution found over all iterations is picked as the result. We also ran it with the same set of parameters as HYBRID, but note that this variant has no use for the set of elite solutions.

The purpose of MULTISTART and MULTISTART+PR is to assess the importance of path-relinking to the overall quality of the algorithm.

The results are summarized in Tables 12.10 (for our algorithm) and 12.11 (for tabu search). For each method and choice of parameters, we present the average percentage

error and the geometric mean of the running times (in seconds). The same information is represented graphically on Figures 12.2 (for GAP) and 12.3 (FPP). Each graph represents the average solution quality as a function of time. They were built directly from Tables 12.10 and 12.11: each line in a table became a point in the corresponding graph, and the points were then linearly interpolated.

Table 12.10 Results on hard classes: average percentage errors and mean running times (in seconds). HYBRID refers to the full algorithm, MULTISTART+PR refers to the algorithm without post-optimization (but with path-relinking between multistart iterations), and MULTISTART refers to the version with no path-relinking at all.

			MULTISTART		MULTISTART+PR		HYBRID	
CLASS	ITER.	ELITE	%ERR	TIME	%ERR	TIME	%ERR	TIME
FPP	4	3	90.60	0.19	86.05	0.39	82.79	0.58
	8	5	83.17	0.31	76.76	0.72	69.37	1.63
	16	7	73.44	0.56	64.19	1.39	53.17	3.41
	32	10	62.10	1.04	50.80	2.76	33.37	7.11
	64	14	51.38	2.01	38.00	5.47	15.84	13.61
	128	20	41.67	3.94	22.23	10.86	4.29	25.15
	256	28	33.69	7.80	9.30	21.68	0.02	47.51
	512	40	25.23	15.50	0.56	43.72	0.01	91.50
	1024	57	14.31	30.89	0.01	88.16	0.00	173.78
	2048	80	4.98	61.72	0.00	177.87	0.00	330.88
GAP	4	3	16.39	0.05	14.11	0.10	12.93	0.14
	8	5	13.59	0.09	11.46	0.19	9.57	0.35
	16	7	11.64	0.17	9.36	0.37	7.47	0.77
	32	10	9.73	0.32	7.62	0.73	5.95	1.63
	64	14	8.14	0.63	6.21	1.45	4.69	3.27
	128	20	7.14	1.26	5.06	2.92	3.83	6.47
	256	28	5.95	2.50	3.75	5.90	2.73	12.52
	512	40	5.00	4.99	2.65	11.92	1.67	24.75
	1024	57	3.81	9.96	1.79	23.62	1.17	48.62
	2048	80	2.77	19.89	1.17	46.57	0.82	92.09

For these series, TABUMS seems to be a better choice than TABU. The difference is particularly noticeable for the FPP instances, where local optima are very far apart. By restarting the tabu search from random solutions at regular intervals, TABUMS is able to explore the search space more effectively than TABU, which relies only on tabu lists to escape local optima. Our method executes even more starts and has the additional advantage of using a more powerful local search. This helps explain why all three variants (HYBRID, MULTISTART+PR and MULTISTART) obtained significanly better results than tabu search, as the pictures show.

Take class GAP. Within 0.25 second, HYBRID can obtain solutions that are more than 10% off optimality (on average); in 20 seconds, the error is down to 2%. The behavior of MULTISTART+PR is almost identical. MULTISTART is slightly better than

Table 12.11 TABU results for hard classes. Two variants are analyzed: TABU starts from a random solution and executes a "pure" tabu search from there; TABUMS runs tabu with until it executes 500 consecutive non-improving iterations, then repeats the procedure from a new random solution. For each method, we show the average percentage error obtained and the mean running times (in seconds).

		TABU		TABUMS	
CLASS	ITER.	AVG%ERR	TIME (S)	AVG%ERR	TIME (S)
FPP	500	95.62	0.63	95.62	0.63
	1000	94.70	1.07	91.69	1.22
	2000	91.03	2.04	87.93	2.43
	4000	86.81	3.82	82.41	4.84
	8000	83.67	7.26	75.04	9.68
	16000	79.32	14.21	66.02	19.47
	32000	75.16	27.45	57.75	38.98
	64000	71.15	52.08	50.77	77.92
GAP	500	15.98	0.26	15.98	0.26
	1000	13.70	0.48	13.21	0.52
	2000	11.66	0.94	11.26	1.04
	4000	10.62	1.67	9.66	2.03
	8000	8.94	3.24	8.07	4.04
	16000	7.72	6.19	7.03	8.10
	32000	7.02	11.75	6.00	16.21
	64000	6.35	22.43	5.06	32.46

both in the beginning, but worse for longer runs, which indicates that path-relinking is important to ensure the robustness of the algorithm. All three variants, however, are better than TABU and TABUMS, whose errors ranged from approximately 16% (in 0.25 second) to around 6% (in 20 seconds).

Even more remarkable differences in performance are observed on class FPP. If given less than one second, all algorithms perform poorly: HYBRID and MULTI-START+PR find solutions that are almost 80% away from optimality on average; TABU and TABUMS are even worse, with 90%; MULTISTART is the best overall, with 70%. However, once longer runs are allowed, HYBRID and (to a lesser degree) MULTI-START+PR improve at a much faster rate than the other methods. Within 50 seconds, HYBRID already finds near-optimal solutions on all cases (the average error is below 0.02%), whereas solutions found by TABUMS are still more than 50% off optimality on average (TABU is even worse, at 70%). Although MULTISTART does significantly better than tabu-based algorithm, it is still at least 5% away from optimality; once again, this shows how important path-relinking is to ensure solution quality.

We stress that such large differences in solution quality are not likely to be observed on "real-world" instances. Classes FPP and GAP are by no means typical, since they were designed with the specific purpose of being hard to solve. For all other classes tested, which have no adversarial structure, the performance of TABU was much closer

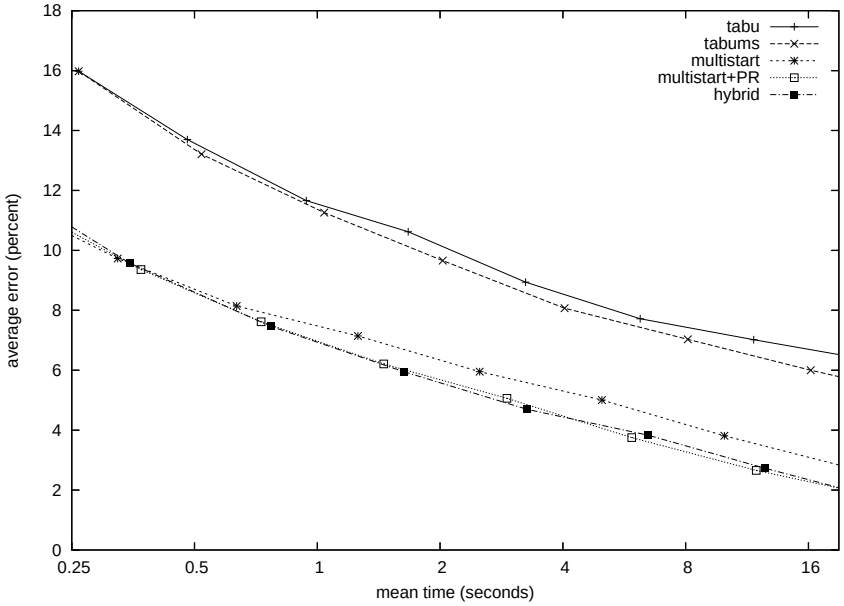


Figure 12.2 GAP class. Mean percent deviation obtained by HYBRID and TABU (and their variants) for several sets of parameters. Data taken from Tables 12.10 and 12.11.

to that of HYBRID. This, however, does not mean the results for FPP and GAP are irrelevant. Practical instances are unlikely to be as hard as those, but nothing guarantees that they will be as well-behaved as the other classes presented here. Since HYBRID is more robust than TABU in extreme cases, it is more likely be faster in moderately difficult ones.

12.4 CONCLUDING REMARKS

We have studied a simple adaptation to the facility location problem of Resende and Werneck's multistart heuristic for the p-median problem (Resende and Werneck, 2004). The resulting algorithm has been shown to be highly effective in practice, finding near-optimal or optimal solutions of a large and heterogeneous set of instances from the literature. In terms of solution quality, the results either matched or surpassed those obtained by some of the best algorithms in the literature on every single class, which shows how robust our method is. The combination of fast local search and path-relinking within a multistart heuristic has proved once again to be a very effective means of finding near-optimal solutions for an NP-hard problem.

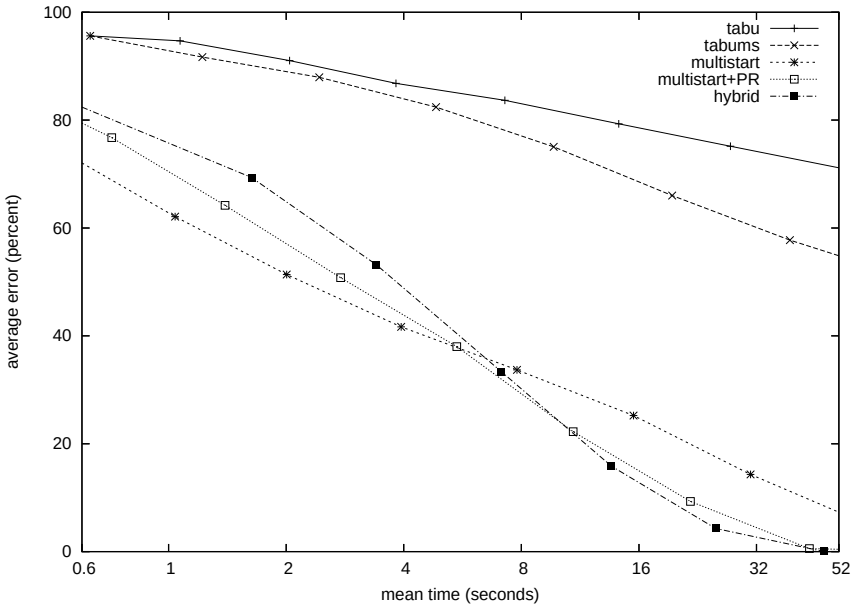


Figure 12.3 FPP class. Average percentage deviation obtained by HYBRID and TABU (and their variants) for several sets of parameters. Data taken from Tables 12.10 and 12.11.

Bibliography

S. Ahn, C. Cooper, G. Cornuéjols, and A. M. Frieze. Probabilistic analysis of a relaxation for the *k*-median problem. *Mathematics of Operations Research*, 13:1–31, 1998.

M. L. Alves and M. T. Almeida. Simulated annealing algorithm for the simple plant location problem: A computational study. *Revista Investigação Operacional*, 12, 1992.

F. Barahona and F. Chudak. Near-optimal solutions to large scale facility location problems. Technical Report RC21606, IBM, Yorktown Heights, NY, USA, 1999.

J. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990. `http://mscmga.ms.ic.ac.uk/info.html`.

J. E. Beasley. Lagrangean heuristics for location problems. *European Journal of Operational Research*, 65:383–399, 1993.

O. Bilde and J. Krarup. Sharp lower bounds and efficient algorithms for the Simple Plant Location Problem. *Annals of Discrete Mathematics*, 1:79–97, 1977.

A. R. Conn and G. Cornueéjols. A projection method for the uncapacited facility location problem. *Mathematical Programming*, 46:273–298, 1990.

G. Cornuéjols, G. L. Nemhauser, and L. A. Wolsey. The uncapacitated facility location problem. In P. B. Mirchandani and R. L. Francis, editors, *Discrete Location Theory*, pages 119–171. Wiley-Interscience, New York, 1990.

J. J. Dongarra. Performance of various computers using standard linear equations software. Technical Report CS-89-85, Computer Science Department, University of Tennessee, 2003.

D. Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26:992–1009, 1978.

R. D. Galvão and L. A. Raggi. A method for solving to optimality uncapacitated facility location problems. *Annals of Operations Research*, 18:225–244, 1989.

D. Ghosh, 2003a. Personal communication.

D. Ghosh. Neighborhood search heuristics for the uncapacitated facility location problem. *European Journal of Operational Research*, 150:150–162, 2003b.

F. Glover. Tabu search and adaptive memory programming: Advances, applications and challenges. In R. S. Barr, R. V. Helgason, and J. L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31:228–248, 1999.

M. Guignard. A Lagrangean dual ascent algorithm for simple plant location problems. *European Journal of Operations Research*, 35:193–200, 1988.

M. Hoefer. Performance of heuristic and approximation algorithms for the uncapacitated facility location problem. Research Report MPI-I-2002-1-005, Max-Planck-Institut für Informatik, 2002a.

M. Hoefer. Ufllib, 2002b. *http://www.mpi-sb.mpg.de/units/ag1/projects/benchmarks/-UflLib/*.

M. Hoefer. Experimental comparision of heuristic and approximation algorithms for uncapacitated facility location. In *Proceedings of the Second International Workshop on Experimental and Efficient Algorithms (WEA 2003)*, number 2647 in Lecture Notes in Computer Science, pages 165–178, 2003.

K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 2003. To appear.

K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *Proc. ACM Symposium on the Theory of Computing (STOC'02)*, 2002.

Y. Kochetov. Benchmarks library, 2003. *http://www.math.nsc.ru/LBRT/k5/Kochetov/-bench.html*.

Y. Kochetov and D. Ivanenko. Computationally difficult instances for the uncapacitated facility location problem. In *Proceedings of the 5th Metaheuristics International Conference (MIC2003)*, 2003.

M. Körkel. On the exact solution of large-scale simple plant location problems. *European Journal of Operational Research*, 39:157–173, 1989.

J. Kratica, D. Tosic, V. Filipovic, and I. Ljubic. Solving the simple plant location problem by genetic algorithm. *RAIRO Operations Research*, 35:127–142, 2001.

A. A. Kuehn and M. J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9(4):643–666, 1963.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *Proceedings of the 5th APPROX Conference*, volume 2462 of *Lecture Notes in Computer Science*, pages 229–242, 2002.

M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

L. Michel and P. Van Hentenryck. A simple tabu search for warehouse location. *European Journal on Operations Research*, 157(3):576–591, 2003.

M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer, 2003.

M. G. C. Resende and C. C. Ribeiro. GRASP with path-relinking: Recent advances and applications. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*. Kluwer, 2005. To appear.

M. G. C. Resende and R. F. Werneck. A fast swap-based local search procedure for location problems. Technical Report TD-5R3KBH, AT&T Labs Research, 2003a.

M. G. C. Resende and R. F. Werneck. On the implementation of a swap-based local search procedure for the p-median problem. In R. E. Ladner, editor, *Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments (ALENEX'03)*, pages 119–127. SIAM, 2003b.

M. G. C. Resende and R. F. Werneck. A hybrid heuristic for the p-median problem. *Journal of Heuristics*, 10(1):59–88, 2004.

D. B. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 265–274, 1997.

13 AN ANNOTATED BIBLIOGRAPHY OF GRASP

Paola Festa[1] and Mauricio G. C. Resende[2]

[2] Departmet of Mathematics and Applications
University of Napoli Federico II
80126 Napoli, Italy
paola.festa@unina.it

[2]Internet and Network Systems Research
AT&T Labs Research
Florham Park, NJ 07932 USA.
mgcr@research.att.com

Abstract: This chapter presents an annotated bibliography of greedy randomized adaptive search procedures (GRASP). The bibliography is current up to early 2004. The bibliography contains: background material; tutorials and surveys; enhancements to the basic method; hybrid methods; software; parallel GRASP; graph theory; quadratic and other assignment problems; location, layout, and cutting; covering, clustering, packing, and partitioning; routing; sequencing and scheduling; logic; manufacturing; transportation; telecommunications; electrical power systems; biology; VLSI design; drawing; and miscellaneous topics. The collection includes papers published in journals, books, M.Sc. and Ph.D. dissertations, and unpublished technical reports. Recent publications are listed in a special section at the end of the chapter. The online version of this bibliography is updated periodically and can be accessed at `http://www.graspheuristic.org`.
Keywords: GRASP, metaheuristics, local search, combinatorial optimization.

13.1 INTRODUCTION

Optimization problems that involve a finite number of alternatives often arise in pratice. Given a finite solution set X and a real-valued function $f : X \to \mathbb{R}$, one seeks a solution $x^* \in X$ with $f(x^*) \leq f(x), \forall\, x \in X$. Common examples include designing efficient telecommunication networks, constructing cost effective airline crew schedules, and producing efficient routes for waste management pickup.

Much work has been done over the last five decades to develop optimal seeking methods that do not explicitly require an examination of each alternative. This research has given rise to the field of *combinatorial optimization* (see Papadimitriou and Steiglitz (1982)), and an increasing capability to solve ever larger real-world problems. Nevertheless, most problems found in practice are either computationally intractable by their nature, or sufficiently large so as to preclude the use of exact algorithms. In such cases, heuristic methods are usually employed to find good, but not necessarily guaranteed optimal solutions. The effectiveness of these methods depends upon their ability to adapt to a particular realization, avoid entrapment at local optima, and exploit the basic structure of the problem. Building on these notions, various heuristic search techniques have been developed that have demonstrably improved our ability to obtain good solutions to difficult combinatorial optimization problems. The most promising of such techniques include simulated annealing (Kirkpatrick, 1984), tabu search (Glover, 1989; 1990; Glover and Laguna, 1997), genetic algorithms (Goldberg, 1989), variable neighborhood search (Hansen and Mladenović, 1998), and GRASP (Greedy Randomized Adaptive Search Procedures) (Feo and Resende, 1989; 1995).

A GRASP is a multi-start or iterative process (Lin and Kernighan, 1973), in which each GRASP iteration consists of two phases, a construction phase, in which a feasible solution is produced, and a local search phase, in which a local optimum in the neighborhood of the constructed solution is sought. The best overall solution is kept as the result.

In the construction phase, a feasible solution is iteratively constructed, one element at a time. The basic GRASP construction phase is similar to the semi-greedy heuristic proposed independently by Hart and Shogan (1987). At each construction iteration, the choice of the next element to be added is determined by ordering all candidate elements (i.e. those that can be added to the solution) in a candidate list C with respect to a greedy function $g : C \rightarrow \mathbb{R}$. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but not necessarily the top candidate. The list of best candidates is called the *restricted candidate list* (RCL).

While local optimization procedures can require exponential time (Johnson et al., 1988) from an arbitrary starting point, empirically their efficiency significantly improves as the initial solution improves. The result is that often many GRASP solutions are generated in the same amount of time required for the local optimization procedure to converge from a single random start. Furthermore, the best of these GRASP solutions is generally significantly better than the single solution obtained from a random starting point.

An especially appealing characteristic of GRASP is the ease with which it can be implemented. Few parameters need to be set and tuned, and therefore development can focus on implementing efficient data structures to assure quick GRASP iterations. Finally, GRASP can be trivially implemented in parallel. Each processor can be initialized with its own copy of the procedure, the instance data, and an independent random

number sequence. The GRASP iterations are then performed in parallel with only a single global variable required to store the best solution found over all processors.

In this article, we provide an annotated bibliography of the GRASP literature up to early 2004. This document contains references related to GRASP that have either appeared in the literature or as technical reports. We first look at tutorials and surveys. Papers that propose enhancements to the basic heuristic are considered next. Following that, we examine GRASP as a component of a hybrid metaheuristic. GRASP source code and parallelization of GRASP follow. The paper concludes with a literature review of operations research and computer science applications of GRASP as well as industrial applications. These include graph theory, quadratic and other assignment problems, location, layout, cutting, covering, clustering, packing, partitioning, routing, sequencing and scheduling, logic, manufacturing, transportation, telecommunications, electrical power systems, biology, VLSI design, drawing, and miscellaneous topics.

This bibliography is updated periodically and can be also accessed at `http://www.graspheuristic.org`. If the reader is aware of any uncited reference, incorrectly cited reference, or update to a cited reference, please contact the authors.

Feo, T. and M. Resende (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters* **8**, 67–71.

This is the first paper to explicitly describe a GRASP. See also page 395.

Feo, T. and M. Resende (1995). Greedy randomized adaptive search procedures. *J. of Global Optimization* **6**, 109–133.

An early survey of GRASP. See also page 353.

Glover, F. (1989). Tabu search – Part I. *ORSA J. on Computing* **1**, 190–206.

Description of the tabu search metaheuristic.

Glover, F. (1990). Tabu search – Part II. *ORSA J. on Computing* **2**, 4–32.

Description of the tabu search metaheuristic.

Glover, F. and M. Laguna (1997). *Tabu Search*. Kluwer Academic Publishers.

Book on metaheuristics and, in particular, tabu search.

Goldberg, D. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.

Book on genetic algorithms.

Hansen, P. and N. Mladenović (1998). An introduction to variable neighborhood search. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol (Eds.), *Metaheuristics, Advances and trends in local search paradigms for optimization*, pp. 433–458. Kluwer Academic Publishers.

Description of the variable neighborhood search metaheuristic.

Hart, J. and A. Shogan (1987). Semi-greedy heuristics: An empirical study. *Operations Research Letters* **6**, 107–114.

Presents a randomized greedy heuristic, called semi-greedy heuristic.

Johnson, D., C. Papadimitriou, and M. Yannakakis (1988). How easy is local search? *Journal of Computer and System Sciences* **17**, 79–100.

Study of the computational complexity of local search.

Kirkpatrick, S. (1984). Optimization by simulated annealing: Quantitative studies. *J. of Statistical Physics* **34**, 975–986.

Description of the simulated annealing metaheuristic.

Lin, S. and B. Kernighan (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research* **21**, 498–516.

An early random multistart local search technique.

Papadimitriou, C. and K. Steiglitz (1982). *Combinatorial optimization: Algorithms and complexity*. Prentice-Hall.

A classic book on combinatorial optimization.

13.2 TUTORIALS AND SURVEYS

Introductory materials, such as tutorials and surveys of GRASP, have appeared in several languages. These papers are listed below.

Bibliography

Blum, C. and A. Roli (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys* **35**(3), 268–308.

> Survey of most important metaheuristics. Differences and similarities among them are highlighted and some advantages and disadvantages of each underlined. Concludes that it is important to design hybrids of metaheuristics.

Feo, T. and M. Resende (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization* **6**, 109–133.

> This tutorial paper defines the various components comprising a GRASP. A general trivial implementation of GRASP on a parallel computer is given. The GRASP literature until 1994 is surveyed. See also page 351.

Festa, P. (2003). Greedy randomized adaptive search procedures. *AIROnews* **7**(4), 7–11.

> The basic components of GRASP and its successful implementations are described. Improved and alternative solution construction mechanisms are discussed along with different techniques for speeding up the search, hybridizations with other metaheuristics, and intensification and post-optimization strategies using path-relinking.

Festa, P. and M. Resende (2002). GRASP: An annotated bibliography. In C. Ribeiro and P. Hansen (Eds.), *Essays and surveys in metaheuristics*, pp. 325–367. Kluwer Academic Publishers.

> An annotated bibliography of GRASP. Covers work related to GRASP that has either appeared in the literature or as technical reports on or before 2001.

González, J. (1996). GRASP. In A. Díaz (Ed.), *Heuristic optimization and neural networks in operations management and engineering*, pp. 143–161. Madrid: Editorial Paraninfo.

> Chapter on GRASP in a book on heuristic procedures for optimization. In Spanish.

Jin-Kao, H., P. Galinier, and M. Habib (2000). Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'Intelligence Artificielle* **13**(2), 283–324.

Highlights main properties of existing metaheuristics to help researchers to choose the most suitable one in practice. One of the metaheuristic addressed is GRASP. In French.

Pitsoulis, L. and M. Resende (2002). Greedy randomized adaptive search procedures. In P. Pardalos and M. Resende (Eds.), *Handbook of Applied Optimization*, pp. 178–183. Oxford University Press.

Chapter with survey of GRASP. Multi-start heuristics are seen as a way to apply local search to solve combinatorial optimization problems. GRASP is shown to, in some ways, improve upon greedy or random multi-start procedures. Includes enhancements to GRASP, parallelization of GRASP, and a survey of GRASP for solving problems in logic, assignment, and location.

Resende, M. (2001). Greedy randomized adaptive search procedures (GRASP). In C. Floudas and P. Pardalos (Eds.), *Encyclopedia of Optimization*, Volume 2, pp. 373–382. Kluwer Academic Publishers.

A survey of GRASP. The basic GRASP is described in detail and enhancements to the basic procedure are given. Several applications of GRASP are reported.

Resende, M. and C. Ribeiro (2003a). GRASP and path-relinking: Recent advances and applications. In T. Ibaraki and Y. Yoshitomi (Eds.), *Proceedings of the Fifth Metaheuristics International Conference (MIC2003)*, pp. T6–1 – T6–6.

The basic components of GRASP and path-relinking, as well as their hybridization, are described. See also page 364.

Resende, M. and C. Ribeiro (2003b). GRASP and path-relinking: Recent advances and applications. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ.

This paper addresses recent advances and application of hybridizations of GRASP and path-relinking. See also page 364.

Resende, M. and C. Ribeiro (2003c). Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger (Eds.), *Handbook of Metaheuristics*, pp. 219–249. Kluwer Academic Publishers.

Survey of GRASP. The basic GRASP is explained in detail and enhancements to the basic procedure are described, including several hybridizations recently proposed in the literature. Several applications of GRASP are reported, showing how this method can find good approximate solutions to operations research problems and industrial applications.

Resende, M. and J. G. Velarde (2003). GRASP: Procedimientos de búsqueda miope aleatorizado y adaptativo (GRASP: Greedy randomized adaptive search procedures). *Inteligencia Artificial* **2**, 61–76.

This paper surveys GRASP. It covers construction mechanisms, local search, the use of path-relinking within GRASP, parallel GRASP, and reviews recent applications of GRASP. In Spanish.

Ribeiro, C. (2002). GRASP: Une métaheuristique gloutone et probabiliste. In J. Teghem and M. Pirlot (Eds.), *Optimisation approchée en recherche opérationnelle*, pp. 153–176. Hermès.

This chapter surveys GRASP. The basic method is described and its applications reviewed. Parallel strategies are discussed and the method's hybridization is considered. In French.

Silveira, C. (1999). GRASP – Uma heuristíca para resolução de problemas de otimização combinatoria. Technical report, Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, RS, Brazil.

This report provides an exhaustive description of the features of GRASP as a metaheuristic method for solving hard combinatorial optimization problems. In Portuguese.

Yagiura, M. and T. Ibakari (1996). Genetic and local search algorithms as robust and simple optimization tools. In *Meta-heuristics: Theory and applications*, pp. 63–82. Kluwer Academic Publishers.

Various metaheuristics such as random multi-start local search (MLS) and genetic algorithm (GA) are implemented and their performance compared. The objective is not to propose the most powerful technique, but to compare general tendencies of various algorithms. From their analysis, they conclude that a GRASP type modification of MLS improves its performance and that GA combined with local search is effective if long computational time is allowed.

13.3 ENHANCEMENTS TO BASIC METHOD

The basic GRASP framework repeatedly builds a greedy randomized solution and applies local improvement to it. Several enhancements to the basic GRASP method have been proposed. Such enhancements can be found in the following papers.

Binato, S. and G. Oliveira (2002). A Reactive GRASP for transmission network expansion planning. In C. Ribeiro and P. Hansen (Eds.), *Essays and surveys in metaheuristics*, pp. 81–100. Kluwer Academic Publishers.

The GRASP previously proposed by Binato, Oliveira, and Araújo (1998) for solving a transmission network expansion problem is enhanced with the reactive scheme of Prais and Ribeiro (2000). See also page 421.

Bresina, J. (1996). Heuristic-biased stochastic sampling. In *Proceedings of the AAAI-96*, pp. 271–278.

Presents a generalization of iterative sampling called Heuristic-Biased Stochastic Sampling (HBSS). HBSS uses a given search heuristic to focus its exploration. The degree of focusing is determined by a chosen *bias function* that reflects the confidence one has in the heuristic's accuracy. This methodology can be directly applied in a GRASP construction phase, by biasing the selection of RCL elements to favor those with higher greedy function values.

Canuto, S., M. Resende, and C. Ribeiro (2001). Local search with perturbation for the prize-collecting Steiner tree problems in graphs. *Networks* **38**, 50–58.

Several local search strategies are investigated for solving the prize-collecting Steiner tree problems in graphs, including path relinking, VNS, and a GRASP that uses cost perturbations. See also page 362, page 374, and page 416.

Fleurent, C. and F. Glover (1999). Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing* **11**, 198–204.

This paper illustrates that constructive multistart methods, such as Random Restart and GRASP, can be improved by the addition of memory and associated heuristic search principles. See also page 382.

Laguna, M. and R. Martí (1999). GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing* **11**, 44–52.

This paper proposes the GRASP with path-relinking hybridization to minimize straight line crossings in a 2-layer graph. See also page 363 and page 427.

Prais, M. (2000). *Estratégias de variação de parâmetros em procedimentos GRASP e aplicações*. Ph. D. thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil.

A GRASP for a matrix decomposition problem in TDMA traffic assignment is described. Proposes Reactive GRASP, a refinement of GRASP where the RCL parameter is adjusted dynamically to favor values that produce good solutions. Reactive GRASP is compared with other RCL strategies on matrix decomposition, set covering, maximum satisfiability and graph planarization.

Prais, M. and C. Ribeiro (2000a). Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing* **12**, 164–176.

This paper describes a GRASP for matrix decomposition problem arising in the context of traffic assignment in communication satellites. A refinement of GRASP, called Reactive GRASP, is proposed. Instead of using a fixed value for the basic parameter that defines the restrictiveness of the candidate list during the construction phase, Reactive GRASP self-adjusts the parameter value according to the quality of the solutions previously found. The local search phase of the GRASP proposed is based on a new neighborhood, defined by constructive and destructive moves. See also page 385 and page 418.

Prais, M. and C. Ribeiro (2000b). Variação de parâmetros em procedimentos GRASP. *Investigación Operativa* **9**, 1–20.

Investigates four strategies for the variation of the GRASP RCL parameter α: 1) reactive – α is self-adjusted along the iterations; 2) uniform – α is randomly chosen from a discrete uniform probability distribution; 3) hybrid – α is randomly chosen from a fixed probability distribution concentrated around the value corresponding to the purely greedy choice; 4) fixed – α has a fixed value close to the purely greedy choice. The reactive strategy is the most flexible and adherent to the characteristics of the specific problem to be solved. In Portuguese.

Rangel, M., N. Abreu, and P. Boaventura Netto (2000). GRASP para o PQA: Um limite de aceitação para soluções iniciais. *Pesquisa Operacional* **20**, 45–58.

Presents a modified version of the GRASP for the quadratic assignment problem proposed by Li, Pardalos, and Resende (1994). The new GRASP uses a criterion to accept or reject a given initial solution, thus trying to avoid potentially fruitless searches. In Portuguese. See also page 385.

Ribeiro, C., E. Uchoa, and R. Werneck (2002). A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing* **14**, 228–246.

A hybrid GRASP is proposed for the Steiner problem in graphs. The construction phase is replaced by either one of several different construction procedures that apply weight perturbation strategies combining intensification and diversification elements. An adaptive path-relinking procedure is used as a post-optimization strategy. See also page 365 and page 378.

13.4 HYBRID METHODS

GRASP has been hybridized with other metaheuristic frameworks, such as tabu search, simulated annealing, genetic algorithms, variable neighborhood search, and path-relinking. The papers in this section propose GRASP hybrids.

Bibliography

Abdinnour-Helm, S. and S. Hadley (2000). Tabu search based heuristics for multi-floor facility layout. *International Journal of Production Research* **38**, 365–383.

Two two-stage heuristics are proposed for solving the multi-floor facility layout problem. GRASP/TS applies a GRASP to find the initial layout and tabu search to refine the initial layout, based on total inter/intra-floor costs. See also page 387.

Ahmadi, S. and I. Osman (2003). Greedy random adaptive memory programming search for the capacitated clustering problem. *European Journal of Operational Research*. To appear.

In order to introduce memory into the pure GRASP framework, the authors propose a GRAMPS framework, which is an hybrid of GRASP and AMP (Adaptive Memory Programming). Local search procedure is descent based on a restricted λ-interchange neighborhood. See also page 393.

Ahuja, R., J. Orlin, and A. Tiwari (2000). A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research* **27**, 917–934.

A genetic algorithm for the QAP is proposed. It incorporates the construction phase of the GRASP for QAP of Li, Pardalos, and Resende (1994) to generate the initial population. See also page 381.

Aiex, R., S. Binato, and M. Resende (2003). Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing* **29**, 393–430.

A parallel GRASP with path-relinking for the job shop problem is described using some ideas proposed in the GRASP of Binato et al. (2002). See also page 369 and page 401.

Aiex, R. and M. Resende (2003). Parallel strategies for GRASP with path-relinking. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ.

This paper analyzes two parallel strategies for GRASP with path-relinking and proposes a criterion to predict parallel efficiency based on experiments with a sequential implementation of the algorithm. The 3-index assignment problem and the job-shop scheduling problem are studied. See also page 369.

Aiex, R., M. Resende, P. Pardalos, and G. Toraldo (2000). GRASP with path relinking for the three-index assignment problem. Technical report, AT&T Labs Research, Florham Park, NJ 07733. To appear in *INFORMS J. on Computing*.

This paper describes variants of GRASP with path relinking for the three index assignment problem (AP3). See also page 381.

Álvarez, A., J. L. González, and K. De-Alba (2003). Scatter search for a network design problem. Technical Report PISIS-2003-02, Universidad Autónoma de Nuevo León, Facultad de Ingeniería Mecánica y Eléctrica, División de Posgrado en Ingeniería de Sistemas, Mexico.

A scatter search (SS) algorithm for solving a fixed charge capacitated multicommodity network design problems on undirected networks is proposed. Different implementations of SS consist in using different subroutines, including a Diversification Generation Method (DGM). The DGM proposed in this paper is a GRASP with some memory features incorporated. See also page 415.

Alvarez-Valdés, R., A. Parajón, and J. Tamarit (2002). A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. *Computers & Operations Research* **29**, 925–947.

This paper proposes several heuristics for the two-dimensional cutting problem. One algorithm is a tabu search procedure, while two further heuristics are simple constructive procedures based on bounds obtained by solving one-dimensional knapsack problems. Those two constructive procedures are embedded in a GRASP. The authors also implement a path-relinking procedure as a post-optimization phase. See also page 387.

Areibi, S. and A. Vannelli (2000). Efficient hybrid search techniques for circuit partitioning. In *IEEE 4th World Multiconference on Circuits, Systems, Communications & Computers*.

Simulated annealing, a tabu search, a GRASP, and a genetic algorithm are applied to circuit partitioning. Two further search techniques are also proposed as hybrids, where a GRASP and a genetic algorithm are used for generating good initial partitions. See also page 388, page 393, and page 425.

Cano, J., O. Cordón, F. Herrera, and L. Sánchez (2002). A GRASP algorithm for clustering. In F. J. Garijo, J. Santos, and M. Toro (Eds.), *Advances in Artificial Intelligence - IBERAMIA 2002, 8th Ibero-American Conference on AI, Seville, Spain, November 12-15, 2002, Proceedings*, Volume 2527 of *Lecture Notes in Computer Science*, pp. 214–223. Springer.

A GRASP for cluster analysis is described. Construction is done using a randomized greedy Kaufman procedure and local search uses the K-means algorithm. The best solutions are found with a hybrid GRASP/K-means with Kaufman initialization. See also page 388 and page 394.

Canuto, S., M. Resende, and C. Ribeiro (2001). Local search with perturbation for the prize-collecting Steiner tree problems in graphs. *Networks* **38**, 50–58.

Local search strategies are investigated, including path relinking, VNS, and a GRASP with cost perturbations. The greedy choice takes into account the input

edge weights, while the local search routine defines as neighborhood of a solution $T(X)$ the set of all minimum spanning trees $T(X')$, where X' differs from X for exactly one node. See also page 357, page 374, and page 416.

Carreto, C. and B. Baker (2002). A GRASP interactive approach to the vehicle routing problem with backhauls. In C. Ribeiro and P. Hansen (Eds.), *Essays and surveys in metaheuristics*, pp. 185–200. Kluwer Academic Publishers.

The incorporation of interactive tools into heuristics is investigated. A GRASP is used in the route construction and improvement phase. Details of the GRASP implementation are given on page 398.

de Souza, M., C. Duhamel, and C. Ribeiro (2003). A GRASP heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy. In M. Resende and J. de Sousa (Eds.), *Metaheuristics: Computer decision-making*, pp. 627–658. Kluwer Academic Publishers.

A novel local search is proposed. It applies reduction techniques to speed up the search and uses a new path-based neighborhood structure, defined by path exchanges. Details are given on page 374.

Delmaire, H., J. Díaz, E. Fernández, and M. Ortega (1999). Reactive GRASP and tabu search based heuristics for the single source capacitated plant location problem. *INFOR* **37**, 194–225.

The single source capacitated plant location problem is a discrete location problem that takes into account capacities in the plants to be opened and imposes that clients be served from a single open plant. A hybrid GRASP is proposed. See also page 389.

Faria Jr., H., S. Binato, M. Resende, and D. F. ao (2004). Power transmission network design by greedy randomized adaptive path relinking. *IEEE Transactions on Power Systems*. To appear.

This papers applies GRASP concepts to path relinking, producing a new metaheuristic (greedy randomized adaptive path relinking) which is applied to solve static power transmission network design problems. See also page 421.

Festa, P., P. Pardalos, M. Resende, and C. Ribeiro (2002). Randomized heuristics for the MAX-CUT problem. *Optimization Methods and Software* **7**, 1033–1058.

The authors propose and test heuristics for the MAX-CUT problem, including a GRASP, a variable neighborhood search (VNS), path relinking, and new hybrid heuristics that combine GRASP, VNS, and path relinking. See also page 375.

Laguna, M. and R. Martí (1999). GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing* **11**, 44–52.

This is the first paper to develop the hybrid GRASP with path relinking. See also page 357 and page 427.

Marinakis, Y. and A. Migdalas (2003). Expanding neighborhood GRASP for the Traveling Salesman Problem. Technical report, Technical University of Crete, Chania 73100, Greece.

A GRASP, that includes tour improvement methods, is proposed for the traveling salesman problem. See also page 376 and page 399.

Oliveira, C., P. Pardalos, and M. Resende (2003). GRASP with path-relinking for the QAP. In T. Ibaraki and Y. Yoshitomi (Eds.), *Proceedings of the Fifth Metaheuristics International Conference (MIC2003)*, pp. 57–1 – 57–6.

A GRASP using path relinking as intensification phase is proposed for the quadratic assignment problem. See also page 383.

Pacheco, J. and O. Valencia (2003). Design of hybrids for the minimum sum-of-squares clustering problem. *Computational Statistics and Data Analysis* **43**, 235–248.

For the non-hierarchical clustering problem under the criterion of minimum sum-of-squares clustering, a set of heuristics are proposed that incorporate genetic operators, local search, and tabu search. See also page 395.

Palubeckis, G. and A. Tomkevicius (2002). GRASP implementations for the unconstrained binary quadratic optimization problem. *Information Technology and Control* **24**, 14–20.

A classical GRASP framework and an enhanced GRASP that uses a simple tabu search as local search are proposed. See also page 434.

Piñana, E., I. Plana, V. Campos, and R. Martí (2004). GRASP and path relinking for the matrix bandwidth minimization. *European J. of Operational Research* **153**(1), 200–210.

Given matrix $A = \{a_{ij}\}_{n\times n}$, the matrix bandwidth minimization problem consists in finding a permutation of the rows and columns of A that keeps the nonzero elements in a band lying as close as possible to the main diagonal of A. The original problem is equivalent to the problem of finding a labeling f of the vertices that minimizes the maximum difference between labels of adjacent vertices. To solve the problem, a GRASP is proposed.

Resende, M. and C. Ribeiro (2003a). GRASP and path-relinking: Recent advances and applications. In T. Ibaraki and Y. Yoshitomi (Eds.), *Proceedings of the Fifth Metaheuristics International Conference (MIC2003)*, pp. T6–1 – T6–6.

Successful implementations of GRASP and path relinking are described in detail and several real world problems application are reported. See also page 354.

Resende, M. and C. Ribeiro (2003b). GRASP and path-relinking: Recent advances and applications. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ.

Recent advances and application of hybridizations of greedy randomized adaptive search procedures (GRASP) and path-relinking are addressed. A template for implementing path-relinking as an intensification procedure for GRASP is presented. Enhancements to the procedure, recently described in the literature, are reviewed. The effectiveness of the procedure is illustrated experimentally. See also page 354.

Resende, M. and C. Ribeiro (2003c). GRASP with path-relinking for private virtual circuit routing. *Networks* **41**, 104–114.

Variants of a GRASP with path-relinking heuristic are proposed for the offline PVC routing problem. See also page 399 and page 419.

Resende, M. and R. Werneck (2003). A hybrid multistart heuristic for the uncapacitated facility location problem. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ.

A multistart heuristic is described based on an idea proposed in Resende and Werneck (2004). See also page 390.

Resende, M. and R. Werneck (2004). A hybrid heuristic for the *p*-median problem. *Journal of Heuristics* **10**, 59–88.

A GRASP with path-relinking with a post-optimization phase is proposed for the *p*-median location problem. See also page 390.

Ribeiro, C., E. Uchoa, and R. Werneck (2002). A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing* **14**, 228–246.

In this hybrid GRASP for the Steiner problem in graphs, the GRASP construction phase is replaced by either one of several different construction procedures that apply weight perturbation strategies combining intensification and diversification elements. An adaptive path-relinking procedure is at the end used as a post-optimization strategy. See also page 358 and page 378.

Ribeiro, C. and D. Vianna (2003). A GRASP/VND heuristic for the phylogeny problem using a new neighborhood structure. Technical report, Department of Computer Science, Catholic U. of Rio de Janeiro, Rio de Janeiro, Brazil.

The authors propose a hybridization of GRASP and VND to solve this problem arising in Biology. The greedy criterion is based on insertions of new taxons into a branch of the current partial solution. The local search phase uses a VND strategy and a new neighborhood structure, called 2-SPR and based on the iterative removing of branches on the current solution. See also page 424.

Souza, M., N. Maculan, and L. Ochi (2003). A GRASP-tabu search algorithm for school timetabling problems. In M. Resende and J. de Sousa (Eds.), *Metaheuristics: Computer decision-making*, pp. 659–672. Kluwer Academic Publishers.

A hybrid approach is proposed that uses a greedy randomized construction phase to obtain a feasible solution to be hopefully improved with tabu search. See also page 405.

13.5 SOFTWARE

This section lists papers that describe computer software for implementing the GRASP framework, as well as codes for solving specific problems.

Bibliography

Andreatta, A., S. Carvalho, and C. Ribeiro (1998). An object-oriented framework for local search heuristics. In *Proceedings of the 26th TOOLS USA '98 – Technology of Object-Oriented Languages and Systems*, pp. 33–45. IEEE Computer Society.

A unified object-oriented framework is proposed for systematically comparing strategies and parameters of different heuristics.

Andreatta, A., S. Carvalho, and C. Ribeiro (2002). A framework for the development of local search heuristics for combinatorial optimization problems. In S. Voss and D. Woodruff (Eds.), *Optimization Software Class Libraries*, pp. 59–79. Kluwer Academic Publishers.

An architectural basis both for the implementation and for the comparison of different local search heuristics is given. Through the use of abstract classes, the framework encapsulates different aspects usually involved in local searches, such as methods for constructing an initial feasible solution, for generating a suitable neighborhood, and for choosing the suitable movement selection criteria.

Festa, P., P. Pardalos, and M. Resende (2001). Algorithm 815: FORTRAN subroutines for computing approximate solution to feedback set problems using GRASP. *ACM Transactions on Mathematical Software* **27**, 456–464.

A set of ANSI standard Fortran 77 subroutines for approximately solving the feedback vertex and arc set problems is described.

Pardalos, P., L. Pitsoulis, and M. Resende (1997). Algorithm 769: Fortran subroutines for approximate solution of sparse quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software* **23**, 196–208.

A version of the GRASP for the quadratic assignment problem of Li, Pardalos, and Resende (1994), tailored for sparse instances is proposed. A set of ANSI standard Fortran 77 subroutines are described. See also page 384.

Resende, M., T. Feo, and S. Smith (1998). Algorithm 787: Fortran subroutines for approximate solution of maximum independent set problems using GRASP. *ACM Transactions on Mathematical Software* **24**, 386–394.

A set of ANSI standard Fortran 77 subroutines for the maximum independent set problem is described. The GRASP used to produce the solutions is proposed in Feo, Resende, and Smith (1994). See also page 378.

Resende, M., P. Pardalos, and Y. Li (1996). Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software* **22**, 104–118.

A set of ANSI standard Fortran 77 subroutines is proposed for dense quadratic assignment problems having at least one symmetric flow or distance matrix. This is an optimized implementation of the algorithm described in Li, Pardalos, and Resende (1994). See also page 385.

Resende, M., L. Pitsoulis, and P. Pardalos (2000). Fortran subroutines for computing approximate solutions of MAX-SAT problems using GRASP. *Discrete Applied Mathematics* **100**, 95–113.

A set of Fortran subroutines for MAX-SAT problems is described. The algorithm implemented was proposed in Resende, Pitsoulis, and Pardalos (1997). See also page 408.

Ribeiro, C. and M. Resende (1999). Algorithm 797: Fortran subroutines for approximate solution of graph planarization problems using GRASP. *ACM Transactions on Mathematical Software* **25**, 341–352.

A set of Fortran subroutines are described, that implements the GRASP for graph planarization of Resende and Ribeiro (1997). See also page 378 and page 429.

13.6 PARALLEL GRASP

GRASP can be easily implemented in parallel. In fact, linear speedup can be expected from a straightforward implementation on independent processors. The following papers deal with parallel GRASP.

Bibliography

Aiex, R. (2002). *Uma investigação experimental da distribuição de probabilidade de tempo de solução em heurísticas GRASP e sua aplicação na análise de implementações paralelas*. Ph. D. thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil.

This thesis describes a new methodology for the analysis of GRASP. Hybrid strategies with path-relinking are also proposed. These are studied on the 3-index assignment problem as well as the job shop scheduling problem and analyzed with the proposed methodology to predict qualitatively how the quality of the solution varies in a parallel independent GRASP. See also page 381 and page 401.

Aiex, R., S. Binato, and M. Resende (2003). Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing* **29**, 393–430.

In this paper, a parallel GRASP with path-relinking as intensification strategy for the job shop problem is described using some ideas proposed in the GRASP of Binato et al. (2002). While in Binato et al. (2002) the greedy function value of an operation k is the makespan resulting from the inclusion of k to the already scheduled operations, in this paper the authors propose the so called *time-remaining* function. It is a greedy function to be used in conjunction with the makespan that favors operations from jobs having long remaining processing times. Two parallelization strategies are proposed: an independent and a cooperative strategy. The independent strategy shows a sub-linear speedup, while the cooperative one shows an approximate linear speedup, thus confirming that the extra time spent for processes communication is compensated by an increase in solution quality. See also page 361 and page 401.

Aiex, R. and M. Resende (2003). Parallel strategies for GRASP with path-relinking. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ.

Independent and cooperative parallel strategies are described and implemented for the 3-index assignment problem and the job-shop scheduling problem. The computational results for independent parallel strategies are shown to qualitatively behave as predicted by the criterion. See also page 361.

Aiex, R., M. Resende, and C. Ribeiro (2002). Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics* **8**, 343–373.

The authors study the probability distributions of solution time to a sub-optimal target value in five GRASPs that have appeared in the literature and for which source code is available. The distributions are estimated by running 12,000 independent runs of the heuristic. Standard methodology for graphical analysis is used to compare the empirical and theoretical distributions and estimate the parameters of the distributions. They conclude that the solution time to a sub-optimal target value fits a two-parameter exponential distribution. Hence, it is possible to approximately achieve linear speed-up by implementing GRASP in parallel.

Alvim, A. (1998). Parallelization strategies for the metaheuristic GRASP. Master's thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, RJ 22453-900 Brazil.

Two parallelization strategies for GRASP are discussed and compared: parallelization by distributing GRASP iterations and parallelization by varying the GRASP random parameter α. Both strategies are adapted to several parallel computation models, such as MPI (Message Passing Interface) and PVM (Parallel Virtual Machine). In Portuguese.

Alvim, A. and C. Ribeiro (1998). Load balancing in the parallelization of the metaheuristic GRASP. Technical report, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, RJ 22453-900 Brazil.

Two parallelization strategies for GRASP are compared. The difference between the two strategies concerns the way in which data is partitioned: pre-scheduled (static load balancing) or self-scheduled (dynamic load balancing). The strategies have been tested considering an application to optimal traffic assignment in TDMA satellite system. Best results have been obtained by using the self-scheduling strategy. In Portuguese.

Catalano, M. F. and F. Malucelli (2000). Parallel randomized heuristics for the set covering problem. Technical report, Transportation and traffic engineering section, Delft U. of Technology, 2600 AG Delft, The Netherlands. To appear in International J. of Computer Research.

Parallel heuristics for set covering are presented. These include a GRASP.

Diaz, J., J. Velázquez, and J. Solis (1999). Un modelo tipo GRASP para la paralelización de algoritmos en computadoras MIMD: Aplicación al algoritmo de Jarmenson. Technical report, Universidad Autónoma del Estado de Morelos, Facultad de Ciencias, Cuernavaca, Morelos, Mexico.

In this paper, the authors use a GRASP to optimize operations needed to parallelize algorithms on a MIMD. The greedy criterion is to minimize the cost associated with communication among processors, while local search uses an exchange neighborhood structure. In Spanish.

Feo, T., M. Resende, and S. Smith (1994). A greedy randomized adaptive search procedure for maximum independent set. *Operations Research* **42**, 860–878.

A GRASP for approximately solving the maximum independent set problem is described. The proposed heuristic can be easily implemented in parallel by decomposing the problem into smaller subproblems, each defined by conditioning

on vertices being in the solution. An implementation of this algorithm was tested on a MIMD computer with up to eight processors. Average linear speedup is observed. See also page 374.

Martins, S. (1999). *Estratégias de Paralelização de Metaheurísticas em Ambientes de Memória Distribuída*. Ph. D. thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil.

This thesis considers parallelization strategies for metaheuristics in distributed memory environments. GRASPs for the Steiner tree problem in graphs are described and implemented in parallel. In Portuguese.

Martins, S., M. Resende, C. Ribeiro, and P. Pardalos (2000). A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization* **17**, 267–283.

A hybrid parallel GRASP for the Steiner problem in graphs is described. See also page 376.

Martins, S., C. Ribeiro, and M. Souza (1998). A parallel GRASP for the Steiner problem in graphs. In A. Ferreira and J. Rolim (Eds.), *Proceedings of IRREGULAR'98 – 5th International Symposium on Solving Irregularly Structured Problems in Parallel*, Volume 1457 of *Lecture Notes in Computer Science*, pp. 285–297. Springer-Verlag.

A parallelization of a sequential GRASP for the Steiner minimal tree problem is proposed. See also page 377.

Murphey, R., P. Pardalos, and L. Pitsoulis (1998). A parallel GRASP for the data association multidimensional assignment problem. In P. Pardalos (Ed.), *Parallel Processing of Discrete Problems*, Volume 106 of *The IMA Volumes in Mathematics and its Applications*, pp. 159–180. Springer-Verlag.

A GRASP for finding good solutions for the data association multidimensional assignment problem is described. The proposed method can be easily parallelized to substantially decrease the running time. See also page 383.

Pardalos, P., L. Pitsoulis, T. Mavridou, and M. Resende (1995). Parallel search for combinatorial optimization: Genetic algorithms, simulated annealing and GRASP. In A. Ferreira and J. Rolim (Eds.), *Parallel Algorithms for Irregularly Structured Problems, Proceedings of the Second International Workshop –Irregular'95*, Volume 980 of *Lecture Notes in Computer Science*, pp. 317–331. Springer-Verlag.

Parallel search techniques for approximating the global optimal solution of combinatorial optimization problems are addressed. For large-scale problems, one of the main limitations of heuristic search is its computational complexity. Efficient parallel implementation of search algorithms can significantly increase the size of the problems that can be solved.

Pardalos, P., L. Pitsoulis, and M. Resende (1995). A parallel GRASP implementation for the quadratic assignment problem. In A. Ferreira and J. Rolim (Eds.), *Parallel Algorithms for Irregularly Structured Problems – Irregular'94*, pp. 115–130. Kluwer Academic Publishers.

Efficient parallel techniques for large-scale sparse quadratic assignment problems are discussed. The GRASP iterations are distributed among the processors.

Each processor is given its own input data and random number sequence and are run independently. A shared global variable stores the value of the incumbent solution. see also page 384.

Pardalos, P., L. Pitsoulis, and M. Resende (1996). A parallel GRASP for MAX-SAT problems. *Lecture Notes in Computer Science* **1184**, 575–585.

A parallel GRASP for weighted maximum satisfiability (MAX-SAT) problem is proposed. The parallel implementation distributes the GRASP iterations among several processors operating in parallel, avoiding that two processors have as input the same random number generator seed. The best solution found among all processors is identified and used as solution of the problem. See also page 407.

Ribeiro, C. and I. Rosseti (2002). A parallel GRASP for the 2-path network design problem. *Lecture Notes in Computer Science* **2004**, 922–926.

A parallel GRASP with path-relinking is proposed for solving 2-path network design problem. See also page 378 and page 419.

Rivera, L. (1998). Evaluation of parallel implementations of heuristics for the course scheduling problem. Master's thesis, Instituto Tecnologico y de Estudios Superiores de Monterrey, Monterrey, Mexico.

This thesis presents several parallel implementations of heuristics for the course scheduling problem. One of the heuristics is a GRASP. In Spanish. See also page 404.

13.7 GRAPH THEORETICAL APPLICATIONS

Perhaps the type of problems where GRASP has been most applied to are problems dealing with graph theory. Below are papers on graph theoretical applications.

Bibliography

Abello, J., P. Pardalos, and M. Resende (1999). On maximum clique problems in very large graphs. In J. Abello and J. Vitter (Eds.), *External memory algorithms and visualization*, Volume 50 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 119–130. American Mathematical Society.

> An approach for clique and quasi-clique computations in very large multigraphs is presented. The authors discuss graph decomposition schemes that break up the original problem into several pieces of manageable dimensions. A semi-external memory GRASP is presented to approximately solve the maximum clique problem and maximum quasi-clique problem. See page 415 for details about GRASP implementation.

Abello, J., M. Resende, and S. Sudarsky (2002). Massive quasi-clique detection. *Lecture Notes in Computer Science* **2286**, 598–612.

> The problem of detecting dense subgraphs (quasi-cliques) in massive sparse graphs whose vertex set fits in RAM is addressed. A GRASP is designed for extracting dense subgraphs in both nonbipartite and bipartite graphs. See also page 415.

Ahuja, R., J. Orlin, and D. Sharma (2001). Multi-exchange neighborhood search structures for the capacitated minimum spanning tree problem. *Mathematical Programming* **91**, 71–97.

> Two very large scale neighborhood search algorithms are proposed for the capacitated minimum spanning tree problem. The first one uses a node based neighborhood structure, defined by performing multi-exchanges involving several trees, where each tree contributes a subtree. The second uses a tree based neighborhood structure, obtained by allowing multi-exchanges, where each tree contributes a subtree. The search algorithms are guided by a GRASP.

Ahuja, R., J. Orlin, and D. Sharma (2003). A composite very large-scale neighborhood structure for the capacitated minimum spanning tree problem. *Operations Research Letters* **31**(3), 185–194.

> The authors propose a composite neighborhood structure that combines both the node based and the tree based neighborhoods already proposed in Ahuja et al.

(2001). They compare their algorithm with the best state-of-art methods, including a GRASP.

Arráiz, E., A. Martínez, O. Meza, and M. Ortega (2001). GRASP and tabu search algorithms for computing the forwarding index in a graph. In *Proceedings of MIC'2001*, pp. 367–370.

The forwarding index in a graph is a measure of the load or of the congestion of vertices once the paths between vertices have been computed. To efficiently compute this number, a GRASP and tabu search is proposed. The GRASP greedy choice considers the length of paths connecting nonadjacent nodes, while two different local search strategies are used. The first replaces a path passing through an internal node with maximum load with another path joining the same endpoints. The second replaces a path of maximum length.

Boeres, M., C. Ribeiro, and I. Bloch (2003). Randomized algorithms for scene recognition by inexact graph matching. Technical report, Computer Science Department, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil.

A new algorithm is proposed to suboptimally solve non-bijective graph matching for model-based pattern recognition. The algorithm consists of a randomized construction procedure and a local search procedure. See also page 431.

Canuto, S., M. Resende, and C. Ribeiro (2001). Local search with perturbation for the prize-collecting Steiner tree problems in graphs. *Networks* **38**, 50–58.

Several local search strategies for the prize collecting Steiner tree problem in graphs are investigated, including path relinking, VNS, and a GRASP with cost perturbations. See also page 357, page 362, and page 416.

Corberán, A., R. Martí, and J. Sanchís (2002). A GRASP heuristic for the mixed Chinese postman problem. *European Journal of Operational Research* **142**, 70–80.

A GRASP for the mixed Chinese postman problem is given. On page 398 details of the implementation of proposed GRASP are given.

de Souza, M., C. Duhamel, and C. Ribeiro (2003). A GRASP heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy. In M. Resende and J. de Sousa (Eds.), *Metaheuristics: Computer decision-making*, pp. 627–658. Kluwer Academic Publishers.

A GRASP for the capacitated minimum spanning tree problem is proposed. It uses a novel local search that considers a new path-based neighborhood structure, defined by path exchanges. The GRASP also makes use of some short term memory elements of tabu search and that implements a randomized version of a savings heuristic in the construction phase. A post-optimization phase is realized by applying a path-relinking procedure. See also page 363.

Feo, T., M. Resende, and S. Smith (1994). A greedy randomized adaptive search procedure for maximum independent set. *Operations Research* **42**, 860–878.

A GRASP for maximum independent set is described. The greedy function chosen orders admissible vertices with respect to the minimum admissible vertex degree. The adjective admissible is referred to a vertex that is not adjacent to any vertex in the current independent set. The neighborhood definition used in the

local search is $(2,1)$-exchange, where two nonadjacent vertices can be added to the current solution if a single vertex from the solution is removed. The proposed heuristic can be easily implemented in parallel by decomposing the problem into smaller subproblems, each defined by conditioning on vertices being in the solution. See also page 370.

Festa, P., P. Pardalos, M. Resende, and C. Ribeiro (2002). Randomized heuristics for the MAX-CUT problem. *Optimization Methods and Software* **7**, 1033–1058.

Randomized heuristics for the MAX-CUT problem are proposed and tested. These include a GRASP and hybrids. See also page 363.

Holmqvist, K., A. Migdalas, and P. Pardalos (1998). A GRASP algorithm for the single source uncapacitated minimum concave-cost network flow problem. In P. Pardalos and D.-Z. Du (Eds.), *Network design: Connectivity and facilities location*, Volume 40 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 131–142. American Mathematical Society.

The single source uncapacitated version of the minimum concave-cost network flow problem is considered. It requires establishing a minimum cost flow through a given network from a single source to a set of sinks. The authors propose a GRASP that can be trivially implemented on parallel processors. The construction phase iteratively builds a tree starting from the source node. The elements of the restricted candidate list are end nodes of arcs with a cost close to the best one. The local search phase applies either of the two local search variants proposed by Guisewite and Pardalos (1990).

Laguna, M. and R. Martí (2001). A GRASP for coloring sparse graphs. *Computational Optimization and Applications* **19**, 165–178.

A GRASP for graph coloring is presented. The GRASP construction phase constructs the next coloring, one color at time. The greedy function chooses the vertex having the maximum degree among the uncolored vertices adjacent to at least one colored vertex. At each step, the local search combines the two smallest cardinality color classes into one and tries to find a valid color for each violating vertex.

Macambira, E. and C. de Souza (1997a). The edge-weighted clique problem: Valid inequalities, facets and polyhedral computations. Technical Report IC-97-14, Instituto de Computação, Universidade Estadual de Campinas, Campinas, Brazil.

The facial structure of the polytope associated with the edge-weighted clique problem is studied and new classes of facets are introduced. To obtain initial lower bounds, a GRASP is used.

Macambira, E. and C. de Souza (1997b). A GRASP for the maximum clique problem with weighted edges. In *Proceedings of the XXIX Brazilian Symposium on Operations Research*, pp. 70.

A branch-and-cut algorithm is proposed for the maximum clique problem with weighted edges. The initialization phase of the algorithm uses a GRASP to generate good starting solutions. The greedy function minimizes the sum of weights of the edges outgoing from vertices in the clique. The local search uses the exchange of one element in the current clique with one not in it.

Macambira, E. and C. Meneses (1998). A GRASP algorithm for the maximum weighted edge subgraph problem. Technical report, Department of Statistics and Computation, University of Ceará, Fortaleza, CE 60740-000 Brazil.

GRASP for the maximum weighted edge subgraph problem is proposed. The greedy function of the construction phase favors the vertices corresponding to the maximum sum of the weights associated with its outgoing edges. The local search tries to improve the actual solution by simply swapping one element in the solution set with one not belonging to the solution. In Portuguese.

Marinakis, Y. and A. Migdalas (2003). Expanding neighborhood GRASP for the Traveling Salesman Problem. Technical report, Technical University of Crete, Chania 73100, Greece.

A GRASP is proposed for the traveling salesman problem using a VNS-like local search. See also page 363 and page 399.

Martí, R. (2001). Arc crossing minimization in graphs with GRASP. *IIE Transactions* **33**, 913–919.

A GRASP for minimizing straight-line crossings in hierarchical graphs is presented. GRASP is shown to be faster than more complex heuristics but produces lower-quality solutions. Though it is not as fast as simple heuristics, it finds better-quality solutions.

Martí, R. and V. Estruch (2001). Incremental bipartite drawing problem. *Computers and Operations Research* **28**, 1287–1298.

The goal of limiting the number of arc crossings is a well accepted criterion of how well a graph is drawn. Incremental graph drawing supports interactive updates by users. A GRASP is proposed for the incremental arc crossing minimization problem for bipartite graphs. See page 427.

Martí, R. and M. Laguna (2003). Heuristics and meta-heuristics for 2-layer straight line crossing minimization. *Discrete Applied Mathematics* **127**, 665–678.

Extensive computational results are presented using 12 heuristics and two metaheuristics for the 2-layer straight line crossing minimization problem. One of the metaheuristics is a GRASP. See page 427.

Martins, S., P. Pardalos, M. Resende, and C. Ribeiro (1999). Greedy randomized adaptive search procedures for the Steiner problem in graphs. In P. Pardalos, S. Rajasekaran, and J. Rolim (Eds.), *Randomization methods in algorithmic design*, Volume 43 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 133–145. American Mathematical Society.

Four versions of a GRASP for approximately solving general instances of the Steiner problem in graphs are proposed. One is implemented and tested. The construction phase is based on the distance network heuristic. The local search is based on insertions and eliminations of nodes to and from the current solution.

Martins, S., M. Resende, C. Ribeiro, and P. Pardalos (2000). A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization* **17**, 267–283.

A GRASP for the Steiner problem in graphs is described. See page 371.

Martins, S., C. Ribeiro, and M. Souza (1998). A parallel GRASP for the Steiner problem in graphs. In A. Ferreira and J. Rolim (Eds.), *Proceedings of IRREGULAR'98 – 5th International Symposium on Solving Irregularly Structured Problems in Parallel*, Volume 1457 of *Lecture Notes in Computer Science*, pp. 285–297. Springer-Verlag.

A parallelization of a sequential GRASP for the Steiner minimal tree problem is proposed. The procedure implemented is one of the procedures described in Martins, Pardalos, Resende, and Ribeiro (1999).

Motta, L., L. Ochi, and C. Martinhon (2001). GRASP metaheuristics to the generalized covering tour problem. In *Proceedings of MIC'2001*, pp. 387–391.

A GRASP for the generalized covering tour problem is presented.

Osman, I., B. Al-Ayoubi, and M. Barake (2003). A greedy random adaptive search procedure for the maximal planar graph problem. *Computers and Industrial Engineering* **45**, 635–651.

A GRASP is proposed and tested for the weighted maximal planar graph problem. GRASP implementation details are given on page 428.

Osman, I., B. Al-Ayoubi, M. Barake, and M. Hasan (2000). A greedy random adaptive search procedure for the weighted maximal planar graph problem. Technical report, School of Business and Center for Advanced Mathematical Sciences, American University of Beirut, Beirut, Lebanon.

A GRASP is proposed and tested for the weighted maximal planar graph problem. Details of the GRASP implementation are given on page 428.

Osman, I., M. Hasan, and A. Abdullah (2002). Linear programming based meta-heuristics for the weighted maximal planar graph. *Journal of the Operational Research Society* **53**, 1142–1149.

Two meta-heuristics are described, including a GRASP, for the weighted maximal planar graph problem. Both are derived from an ILP relaxation. See page 428.

Pardalos, P., T. Qian, and M. Resende (1999). A greedy randomized adaptive search procedure for the feedback vertex set problem. *Journal of Combinatorial Optimization* **2**, 399–412.

A GRASP is described for the feedback vertex set problem on a digraph. Several greedy functions are tested, all of them taking into account vertices with high degree. The local search procedure tries at each iteration to eliminate redundant vertices. Some efficient problem reduction techniques are also described. They are useful both to simplify the problem instance and to determine whether a digraph is acyclic.

Pardalos, P., M. Resende, and J. Rappe (1998). An exact parallel algorithm for the maximum clique problem. In R. D. Leone, A. Murli, P. Pardalos, and G. Toraldo (Eds.), *High performance algorithms and software in nonlinear optimization*, pp. 279–300. Kluwer Academic Publishers.

A parallelized version of the exact algorithm of Carraghan and Pardalos (1990) for the unweighted maximum clique problem is described. A variant of the

GRASP for the maximum independent set problem of Feo, Resende, and Smith (1994) is used for computing feasible solutions.

Resende, M., T. Feo, and S. Smith (1998). Algorithm 787: Fortran subroutines for approximate solution of maximum independent set problems using GRASP. *ACM Transactions on Mathematical Software* **24**, 386–394.

A set of ANSI standard Fortran 77 subroutines to find an approximate solution of a maximum independent set problem using the GRASP of Feo, Resende, and Smith (1994). See page 367.

Resende, M. and C. Ribeiro (1997). A GRASP for graph planarization. *Networks* **29**, 173–189.

A GRASP for the graph planarization problem is described in detail, extending the two-phase heuristic of Goldschmidt and Takvorian (*Networks*, v. 24, pp. 69–73, 1994). See also page 428.

Ribeiro, C. and M. Resende (1999). Algorithm 797: Fortran subroutines for approximate solution of graph planarization problems using GRASP. *ACM Transactions on Mathematical Software* **25**, 341–352.

A set of Fortran subroutines that implements the GRASP for graph planarization of Resende and Ribeiro (1997) is described. See also page 368 and page 429.

Ribeiro, C. and I. Rosseti (2002). A parallel GRASP for the 2-path network design problem. *Lecture Notes in Computer Science* **2004**, 922–926.

A parallel GRASP with path-relinking is proposed for finding approximate solutions to the 2-path network design problem. See also page 372 and page 419.

Ribeiro, C., E. Uchoa, and R. Werneck (2002). A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing* **14**, 228–246.

In this hybrid GRASP, an adaptive path-relinking procedure is used as a post-optimization strategy. The GRASP construction phase is replaced by either one of several different construction procedures that apply weight perturbation strategies combining intensification and diversification elements. The local search phase circularly explores two different strategies. The first defines a simple node-based neighborhood structure. The second one uses a key-path-based neighborhood, where a key-node is a Steiner node with degree at least three and a key-path is a Steiner tree T whose extremities are either terminals of key-node (if there are any, intermediate nodes are Steiner node with degree two in T). See also page 358 and page 365.

Rosseti, I. (2003). *Heurísticas para o problema de síntese de redes a 2-caminhos*. Ph. D. thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil.

This thesis proposes sequential and parallel heuristics for the 2-path network design problem. These include variants and combinations of GRASP. In Portuguese. See also page 419.

13.8 QUADRATIC AND OTHER ASSIGNMENT PROBLEMS

GRASP has been applied to the quadratic assignment problems, as well as other assignment problems, such as biquadratic, three index, and multidimensional assignment. The following papers fall into this category.

Bibliography

Ahuja, R., J. Orlin, and A. Tiwari (2000). A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research* **27**, 917–934.

A genetic algorithm for the QAP that incorporates the construction phase of the GRASP for QAP of Li, Pardalos, and Resende (1994) to generate the initial population is described. See also page 361.

Aiex, R. (2002). *Uma investigação experimental da distribuição de probabilidade de tempo de solução em heurísticas GRASP e sua aplicação na análise de implementações paralelas*. Ph. D. thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil.

This thesis describes a new methodology for the analysis of GRASP. Hybrid strategies with path-relinking are also proposed. These are studied on the 3-index assignment problem as well as the job shop scheduling problem. See also page 369 and page 401.

Aiex, R., M. Resende, P. Pardalos, and G. Toraldo (2000). GRASP with path relinking for the three-index assignment problem. Technical report, AT&T Labs Research, Florham Park, NJ 07733. To appear in *INFORMS J. on Computing*.

Variants of GRASP with path relinking for the three index assignment problem (AP3) are presented. Computational results show clearly that this GRASP for AP3 benefits from path relinking and that the variants considered in this paper compare well with previously proposed heuristics for this problem. The authors also show that the random variable time to target solution, for all proposed GRASP with path relinking variants, fits a two-parameter exponential distribution. To illustrate the consequence of this, one of the variants of GRASP with path relinking is shown to benefit from parallelization. See also page 362.

Bard, J. (1997). An analysis of a rail car unloading area for a consumer products manufacturer. *Journal of the Operational Research Society* **48**, 873–883.

This paper discusses how to design and analyze the rail-car unloading area of Proctor & Gamble's principal laundry detergent plant. The related combinatorial problem of assigning rail-cars to positions on the platform and unloading equipment to rail-cars is modeled as a mixed-integer nonlinear program. See also page 411.

Feo, T. and J. González-Velarde (1995). The intermodal trailer assignment problem: Models, algorithms, and heuristics. *Transportation Science* **29**, 330–341.

The problem of optimally assigning highway trailers to rail-car hitches in intermodal transportation is addressed. Using a set covering formulation, the problem is modeled as an integer linear program, whose linear programming relaxation yields a tight lower bound. This formulation also provides the basis for developing a branch-and-bound algorithm and a GRASP for solving the problem. See also page 412.

Fleurent, C. and F. Glover (1999). Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing* **11**, 198–204.

This paper illustrates that constructive multistart methods, such as Random Restart and GRASP, can be improved by the addition of memory and associated heuristic search principles. It shows that the GRASP for QAP of Li, Pardalos, and Resende (1994) can be improved upon by using these memory strategies. See also page 357.

Li, Y., P. Pardalos, and M. Resende (1994). A greedy randomized adaptive search procedure for the quadratic assignment problem. In P. Pardalos and H. Wolkowicz (Eds.), *Quadratic assignment and related problems*, Volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 237–261. American Mathematical Society.

A GRASP for the quadratic assignment problem is described. Construction first makes two assignments, and then completes the solution by making assignments, one at a time. The greedy function is assignment interaction cost. The local search procedure is a 2 assignment exchange.

Liu, X., P. Pardalos, S. Rajasekaran, and M. Resende (2000). A GRASP for frequency assignment in mobile radio networks. In S. Rajasekaran, P. Pardalos, and F.Hsu (Eds.), *Mobile Networks and Computing*, Volume 52 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 195–201. American Mathematical Society.

A GRASP for frequency assignment is described. Local search uses simulated annealing. See also page 417.

Lourenço, H. R. and D. Serra (1998). Adaptive approach heuristics for the generalized assignment problem. Technical Report 288, Department of Economics and Business, Universitat Pompeu Fabra, Barcelona, Spain.

Several heuristics based a hybrid approach are proposed. This includes a GRASP, whose greedy choice is to prefer tasks using small amounts of agent resource. In the improvement phase, several local searches are implemented, among them a descent local search and a search based on ejection chain neighborhood.

Mavridou, T., P. Pardalos, L. Pitsoulis, and M. Resende (1998). A GRASP for the biquadratic assignment problem. *European Journal of Operational Research* **105**, 613–621.

A GRASP for the biquadratic assignment problem is proposed. The construction phase has two stages. The first stage simultaneously makes four assignments, selecting the pairs corresponding to the smallest interaction costs, while the second stage makes the remaining assignments, one at time. The greedy function in the second stage selects the assignment corresponding to the minimum interaction cost with respect to the already-made assignments. In the local search phase, 2-exchange local search is applied to the permutation constructed in the first phase.

Murphey, R., P. Pardalos, and L. Pitsoulis (1998a). A greedy randomized adaptive search procedure for the multitarget multisensor tracking problem. In P. Pardalos and D.-Z. Du (Eds.), *Network design: Connectivity and facilities location*, Volume 40 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 277–301. American Mathematical Society.

A GRASP is presented for the multitarget multisensor tracking problem, which can be interpreted as a collection of multidimensional assignment problems. A likelihood cost function and partitioning constraint set are developed. The GRASP construction phase creates the restricted candidate list containing the most likely to occur (lower cost) tuples. The local search explores all 2-exchange permutations.

Murphey, R., P. Pardalos, and L. Pitsoulis (1998b). A parallel GRASP for the data association multidimensional assignment problem. In P. Pardalos (Ed.), *Parallel Processing of Discrete Problems*, Volume 106 of *The IMA Volumes in Mathematics and its Applications*, pp. 159–180. Springer-Verlag.

A parallel GRASP for the data association multidimensional assignment problem is described. At each discrete time interval, the data set is formulated as a multidimensional assignment problem (MAP) with a maximum likelihood cost function. A near-optimal solution to each MAP is obtained with a GRASP. See also page 371.

Oliveira, C. and F. Gomes (1999). Two metaheuristics for channel allocation in mobile telephony. Technical report, Artificial Intelligence Laboratory, Universidade Federal do Ceará, Fortaleza, Brazil.

This frequency assignment problem consists in minimizing the total system interference in mobile phone covered areas, with respect to co-channel and adjacent-channel interference. Two metaheuristics are proposed: GRASP and Asynchronous Team (A-Team). See also page 418.

Oliveira, C., P. Pardalos, and M. Resende (2003). GRASP with path-relinking for the QAP. In T. Ibaraki and Y. Yoshitomi (Eds.), *Proceedings of the Fifth Metaheuristics International Conference (MIC2003)*, pp. 57–1 – 57–6.

In this paper, a GRASP using path relinking as intensification phase is proposed for the quadratic assignment problem. The GRASP is based on the GRASP proposed in Li et al. (1994). See also page 364.

Oliveira, C., P. Pardalos, and M. Resende (2004). GRASP with path-relinking the the quadratic assignment problem. *Lecture Notes in Computer Science*. To appear.

This paper describes a GRASP for quadratic assignment problem which uses a faster local search than in Li, Pardalos, and Resende (1994) and has a path-relinking intensification component. Computational results show that GRASP is sped up considerably when used in conjunction with the path-relinking component.

Pardalos, P., L. Pitsoulis, and M. Resende (1995). A parallel GRASP implementation for the quadratic assignment problem. In A. Ferreira and J. Rolim (Eds.), *Parallel Algorithms for Irregularly Structured Problems – Irregular'94*, pp. 115–130. Kluwer Academic Publishers.

Efficient parallel techniques for large-scale sparse quadratic assignment problems are discussed. The paper provides a detailed description of a parallel implementation on an MIMD computer of the sequential GRASP proposed by Li, Pardalos, and Resende (1994) for solving the QAP. See also page 371.

Pardalos, P., L. Pitsoulis, and M. Resende (1997). Algorithm 769: Fortran subroutines for approximate solution of sparse quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software* **23**, 196–208.

A version of the GRASP for the quadratic assignment problem of Li, Pardalos, and Resende (1994), tailored for sparse instances is proposed. A set of ANSI standard Fortran 77 subroutines are described. See also page 367.

Pardalos, P., K. Ramakrishnan, M. Resende, and Y. Li (1997). Implementation of a variance reduction based lower bound in a branch and bound algorithm for the quadratic assignment problem. *SIAM Journal on Optimization* **7**, 280–294.

A branch & bound algorithm for the quadratic assignment problem if proposed. It uses a GRASP to compute upper bounds.

Pasiliao, E. (1998). A greedy randomized adaptive search procedure for the multi-criteria radio link frequency assignment problem. Technical report, Department of ISE, University of Florida, Gainesville, FL 32611-6595.

A GRASP for computing approximate solutions to a radio link frequency assignment problem is proposed. The objective is to minimize the order and the span of the solution set. See also page 418.

Pitsoulis, L. (1999). *Algorithms for nonlinear assignment problems*. Ph. D. thesis, Department of Industrial and Systems Engineering, University of Florida.

This dissertation presents GRASPs for solving the following NP-hard nonlinear assignment problems (NAPs): quadratic assignment problem (QAP), biquadratic assignment problem (BiQAP), turbine balancing problem (TBP), and multidimensional assignment problem (MAP). Computational results indicate that all of the suggested algorithms are among the best in the literature in terms of solution quality and computational time.

Pitsoulis, L., P. Pardalos, and D. Hearn (2001). Approximate solutions to the turbine balancing problem. *European J. of Operational Research* **130**, 147–155.

The turbine balancing problem is formulated as a standard quadratic assignment problem, and a GRASP for solving the resulting problem is presented.

Prais, M. and C. Ribeiro (2000). Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing* **12**, 164–176.

A GRASP is described for a matrix decomposition problem arising in the context of traffic assignment in communication satellites. A geostationary communication satellite has a number of spot beam antennas covering geographically distributed areas. According to the slot switching configuration on the on-board switch, the uplink traffic received at the satellite has to be immediately sent to ground areas through a set of transponders. The slot switching configurations are determined through the solution of a time slot assignment problem, which is equivalent to the decomposition of a nonnegative traffic matrix into the sum of a family of switching mode matrices. A refinement of GRASP, called Reactive GRASP, is proposed. See also page 358 and page 418.

Rangel, M., N. Abreu, and P. Boaventura Netto (2000). GRASP para o PQA: Um limite de aceitação para soluções iniciais. *Pesquisa Operacional* **20**, 45–58.

A modified version of the GRASP proposed in Li, Pardalos, and Resende (1994) for the quadratic assignment problem is presented. It computes a normalized limit cost, defined with the aid of QAP upper and lower bounds easily obtained and discards all solutions with cost less than the computed limit. In Portuguese. See also page 358.

Rangel, M., N. de Abreu, P. Boaventura Netto, and M. Boeres (1998). A modified local search for GRASP in the quadratic assignment problem. Technical report, Production Engineering Program, COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, RJ Brazil.

An improvement of the local search phase of the GRASP proposed by Li, Pardalos, and Resende (1994) for solving the quadratic assignment problem is proposed. The new strategy amplifies the local search range and improves the local search's efficiency.

Resende, M., P. Pardalos, and Y. Li (1996). Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software* **22**, 104–118.

A set of ANSI standard Fortran 77 subroutines is described for dense quadratic assignment problems having at least one symmetric flow or distance matrix. It is an optimized implementation of the algorithm described in Li, Pardalos, and Resende (1994). See also page 368.

Robertson, A. (2001). A set of greedy randomized adaptive local search procedure (GRASP) implementations for the multidimensional assignment problem. *Computational Optimization and Applications* **19**, 145–164.

Four GRASP implementations are introduced for the multidimensional assignment problem. They combine two constructive methods (randomized greedy and randomized max regret) and two local search methods (2-exchange and variable depth exchange). The greedy function of the randomized max regret construction method is a measure of the competition between the two leading cost candidates. The maximum regret value corresponds to the candidate assignment that has the

largest winning margin between itself and its next highest competitor. The variable depth exchange is an extension of the 2-exchange method that allows a more extensive search of the surrounding neighborhood space.

Sosnowska, D. (2000). Optimization of a simplified fleet assignment problem with metaheuristics: Simulated annealing and GRASP. In P. Pardalos (Ed.), *Approximation and complexity in numerical optimization*. Kluwer Academic Publishers.

Two heuristics based on simulated annealing and GRASP are presented for a simplified fleet assignment problem. Both methods are based on swapping parts of sequence of flight legs assigned to an aircraft (rotation cycle) between two randomly chosen aircrafts. See also page 413.

Urban, T., W.-C. Chiang, and R. Russel (2000). The integrated machine allocation and layout problem. *International Journal of Production Research* **38**, 2911–2930.

GRASP is used to solve quadratic assignment sub-problems in a model that aggregates quadratic assignment problems with several network flow problems with side constraints. This model is used to produce machine layouts where machines are not required to be placed in a functional or cellular layout. See also page 391.

Vieira, C. and P. Gondim (2001). Uma nova estratégia para aplicação do GRASP ao problema de alocação de canal. Technical Report 070/DE9/01, Departamento de Engenharia de Sistemas, Instituto Militar de Engenharia, Rio de Janeiro, Brazil.

A special location problem arising in telecommunications is addressed. A heuristic that uses the GRASP randomized construction idea is proposed. The chosen greedy function measures the difficulty in assigning a frequency. See also page 420.

13.9 LOCATION AND LAYOUT

Location and layout are another class of problems successfully handled by GRASP. The following papers show how GRASP is used in this context.

Abdinnour-Helm, S. and S. Hadley (2000). Tabu search based heuristics for multi-floor facility layout. *International Journal of Production Research* **38**, 365–383.

A GRASP/Tabu Search is proposed applying a GRASP to find the initial layout and tabu search to refine the layout. Computational tests indicate that GRASP/Tabu Search compares favorably with other heuristics. See also page 361.

Alvarez-Valdes, R., A. Parajon, and J. Tamarit (2001). A computational study of heuristic algorithms for two-dimensional cutting stock problems. In *Proceedings of MIC'2001*, pp. 7–11.

To solve the column generation subproblem arising while solving two-dimensional cutting stock problem, a Gilmore-Gomory approach and several heuristics, including a GRASP, are proposed. The greedy criterion is the potential benefit of an item, while at each iteration of the local search, waste rectangles produced in the construction phase are merged if possible with adjacent pieces, creating new rectangles that can be cut with greater value.

Alvarez-Valdés, R., A. Parajón, and J. Tamarit (2002). A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. *Computers & Operations Research* **29**, 925–947.

Several heuristics and hybridizations are proposed for the two-dimensional cutting problem, in which a single stock sheet has to be cut into a set of small pieces, while maximizing the value of the pieces cut. See page 362.

Amaldi, E., A. Capone, and F. Malucelli (2003). Planning UMTS base station location: Optimization models with power control and algorithms. *IEEE Transactions on Wireless Communications* **2**(5), 939–952.

Two greedy randomized algorithms and a tabu search are proposed for the UMTS base station location problem. See also page 415.

Amaldi, E., A. Capone, F. Malucelli, and F. Signori (2003). Optimization models and algorithms for downlink UMTS radio planning. In *Wireless Communications and Networking, 2003 (WCNC 2003)*, Volume 2, pp. 827–831.

This paper deals with the UMTS base station location problem. Two mathematical programming formulations are provided for locating directive base stations considering downlink (base station to mobile) direction and assuming a power-based as well or a SIR-based (Signal-to-Interface Ratio) power control mechanism. A GRASP is used to solve the problem. See also page 416.

Areibi, S. (1999). GRASP: An effective constructive technique for VLSI circuit partitioning. In *Proc. IEEE Canadian Conference on Electrical & Computer Engineering (CCECE'99)*, Volume 1, pp. 462–467.

A GRASP is proposed for producing good initial solutions for an iterative improvement technique. At each iteration of the randomized approach, the gains associated with moving modules to the current block being filled are examined, and a restricted candidate list is built using the modules with the highest gains. See also page 393 and page 425.

Areibi, S. and A. Vannelli (1997). A GRASP clustering technique for circuit partitioning. In J. Gu and P. Pardalos (Eds.), *Satisfiability problems*, Volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 711–724. American Mathematical Society.

A basic node interchange scheme is proposed for solving the circuit partitioning problem. A clustering technique that uses GRASP to generate clusters of moderate sizes is described. Details of the GRASP implementation are given on page 425.

Areibi, S. and A. Vannelli (2000). Efficient hybrid search techniques for circuit partitioning. In *IEEE 4th World Multiconference on Circuits, Systems, Communications & Computers*.

Simulated annealing, tabu search, GRASP, and a genetic algorithm are proposed for circuit partitioning. Two further search techniques are also proposed as hybrids, where a GRASP and a genetic algorithm are used for generating good initial partitions. See also page 362, page 393, and page 425.

Bautista, J. and J. Pereira (2003). Procedimientos para la localización de áreas de aportación de residuos urbanos. In *27 Congreso Nacional de Estadística e Investigación Operativa*, Lleida, Spain.

An exact method, a genetic algorithm, and a GRASP are proposed to solve the problem of locating areas where to collect waste products. The two metaheuristics share the same local search strategy that simply looks for redundant elements to be removed from the solution. In Spanish.

Cano, J., O. Cordón, F. Herrera, and L. Sánchez (2002). A GRASP algorithm for clustering. In F. J. Garijo, J. Santos, and M. Toro (Eds.), *Advances in Artificial Intelligence - IBERAMIA 2002, 8th Ibero-American Conference on AI, Seville, Spain, November 12-15, 2002, Proceedings*, Volume 2527 of *Lecture Notes in Computer Science*, pp. 214–223. Springer.

A GRASP for cluster analysis is described. See also page 362 and page 394.

Chiang, W., P. Kouvelis, and T. Urban (2002). Incorporating workflow interference in facility layout design: The quartic assignment problem. *Management Science* **48**(4), 584–590.

A branch & bound and a tabu search are proposed for the quartic assignment problem. Computational efficiency and performance of the proposed methods are investigated on a set of randomly generated instances by comparing them with the GRASP for the biquadratic assignment problem proposed in Mavridou et al. (1998).

Delmaire, H., J. Díaz, E. Fernández, and M. Ortega (1997). Comparing new heuristics for the pure integer capacitated plant location problem. Technical Report DR97/10, Department of Statistics and Operations Research, Universitat Politecnica de Catalunya, Barcelona, Spain.

Several heuristics are proposed to solve the pure integer capacitated plant location problem: evolutionary algorithms, GRASP, simulated annealing, and tabu search. All the algorithms share the same neighborhood definition.

Delmaire, H., J. Díaz, E. Fernández, and M. Ortega (1999). Reactive GRASP and tabu search based heuristics for the single source capacitated plant location problem. *INFOR* **37**, 194–225.

A reactive GRASP is embedded in a tabu search algorithm for the single source capacitated plant location problem. See also page 363.

García, J. D. (2001). *Algorithmic approaches for the single source capacitated plant location problem*. Ph. D. thesis, Departamento d'Estadística i Investigació Operativa, Univarsitat Politécnica de Catalunya, Barcelona, Spain.

This thesis proposes several algorithmic alternatives based on both exact and approximate methods for efficiently solving the single source capacitated plant location problem. One of the proposals is a reactive GRASP that uses a greedy function based on a percentage of the sum of the cost associated with opening a plant and the cost of allocating clients. The local search procedure uses two neighborhood structures: a client shift neighborhood and a client swap neighborhood.

Gomes, M. and J. da Silva (1999). An experimental evaluation of the GRASP metaheuristic applied to the uncapacitated location problem. Technical Report 004/99, Department of Statistics and Computation, State University of Ceará, Fortaleza, Ceará, Brazil.

Two GRASP heuristics, one using the ADD heuristic and the other using the DROP heuristic, are proposed for the uncapacitated location problem. Computational experiments with instances from Beasley's OR-Library show that GRASP-DROP dominates GRASP-ADD, while both GRASP heuristics dominate ADD and DROP.

Holmqvist, K., A. Migdalas, and P. Pardalos (1997). Greedy randomized adaptive search for a location problem with economies of scale. In I. B. et al. (Ed.), *Developments in Global Optimization*, pp. 301–313. Kluwer Academic Publishers.

A GRASP is proposed for finding approximate solutions to a facility location problem with concave costs. The greedy function of the construction phase favors the facilities that give lower cost for a costumer, regarding the effect that already connected costumers have on the solution. The neighborhood function is defined as changing facility connection for one costumer. Instead of a time

consuming computation of the objective function value for each neighborhood solution, the difference in cost for changing supplier is examined.

Klincewicz, J. (1992). Avoiding local optima in the *p*-hub location problem using tabu search and GRASP. *Annals of Operations Research* **40**, 283–302.

Two heuristics are proposed, based on tabu search and GRASP, for the *p*-hub location problem. The objective is to overcome the difficulty that local search algorithms encounter. See also page 417.

Klincewicz, J. (2002). Enumeration and search procedures for a hub location problem with economies of scale. *Annals of Operations Research* **110**, 107–122.

An optimal enumeration scheme, as well as other heuristics based on tabu search and GRASP are proposed for locating hubs in a communications or transportation network. See also page 417.

Kumaran, K., A. Srinivasan, Q. Wang, S. Lanning, and K. Ramakrishnan (2001). Efficient algorithms for location and sizing problems in network design. In *Global Telecommunications Conference, 2001 (GLOBECOM '01)*, Volume 4, pp. 2586–2590. IEEE.

Algorithms based on linear programming and a slight modified GRASP are developed. The construction phase is performed at random. Given an initial solution, the local search procedure exhaustively evaluate objective function value for all possible single location changes. See also page 417.

Resende, M. and R. Werneck (2003). A hybrid multistart heuristic for the uncapacitated facility location problem. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ.

A multistart heuristic is proposed for the uncapacitated facility location problem, based on an idea proposed for the *p*-median problem in Resende and Werneck (2004). The algorithm consists in two phases. The first phase is a multistart procedure that builds a randomized solution from which to apply a local search routine and a path-relinking. The second is a post-optimization phase realized by applying path-relinking over the whole elite set. See also page 365.

Resende, M. and R. Werneck (2004). A hybrid heuristic for the *p*-median problem. *Journal of Heuristics* **10**, 59–88.

A GRASP with path-relinking as intensification and post-optimization phase is proposed for the *p*-median problem. See also page 365.

Serra, D. and R. Colomé (2001). Consumer choice in competitive location models: Formulations and heuristics. *Papers in Regional Science* **80**, 439–464.

The importance of customer behavior with respect to distance or transportation costs in the optimality of locations obtained by traditional state-of-the-art competitive location models is addressed. Four models to represent the problem are proposed. A hybrid metaheuristic is proposed for solving it.

Silva, F. and D. Serra (2002). Locating emergency services with priority rules: The priority queuing covering location problem. Technical Report 1, Research group in management logistics, Department of Economics and Business, Universitat Pompeu Fabra, Barcelona, Spain.

The greedy criterion used by the GRASP proposed is based on distances. For each server, one at a time, the local search iteratively de-allocates all demands allocated to it and moves them to all possible unused candidates.

Urban, T. (1998). Solution procedures for the dynamic facility layout problem. *Annals of Operations Research*, 323–342.

The concept of incomplete dynamic programming is applied to the dynamic facility layout problem and a lower bound for the general problem is developed. A GRASP and an initialized multi-greedy algorithm are described to provide a solution methodology for large problems. The GRASP is the algorithm proposed by Li, Pardalos, and Resende (1994) for dense quadratic assignment problems.

Urban, T., W.-C. Chiang, and R. Russel (2000). The integrated machine allocation and layout problem. *International Journal of Production Research* **38**, 2911–2930.

GRASP is used to solve quadratic assignment sub-problems in a model that aggregates quadratic assignment problems with several network flow problems with side constraints. See also page 386.

13.10 COVERING, CLUSTERING, PACKING, AND PARTITIONING

Covering, clustering, packing, and partitioning are problem areas where GRASP has been recently successfully applied. The papers below illustrate this.

Bibliography

Ahmadi, S. and I. Osman (2003). Greedy random adaptive memory programming search for the capacitated clustering problem. *European Journal of Operational Research*. To appear.

A GRAMPS framework, which is a hybrid of GRASP and AMP (Adaptive Memory Programming) is proposed for the capacitated clustering problem. See also page 361.

Areibi, S. (1999). GRASP: An effective constructive technique for VLSI circuit partitioning. In *Proc. IEEE Canadian Conference on Electrical & Computer Engineering (CCECE'99)*, Volume 1, pp. 462–467.

At each iteration of the proposed randomized approach, the gains associated with moving modules to the current block being filled are examined and a restricted candidate list is built using the modules with the highest gains. See also page 388 and page 425.

Areibi, S. and A. Vannelli (2000). Efficient hybrid search techniques for circuit partitioning. In *IEEE 4th World Multiconference on Circuits, Systems, Communications & Computers*.

Some pure metaheuristics and as well as heuristic hybridizations are proposed for circuit partitioning. A GRASP and a genetic algorithm are used for generating good initial partitions. See also page 362, page 388, and page 425.

Argüello, M., T. Feo, and O. Goldschmidt (1996). Randomized methods for the number partitioning problem. *Computers & Operations Research* **23**(2), 103–111.

Randomized methodologies for solving the number partitioning problem are proposed. The greedy criterion consists in considering only large elements for differencing. Specific selection of the elements to be differenced is made at random. Differences are placed back into the list of remaining elements, and the process of selecting the next element is repeated. The proposed methods are greedy, randomized, and adaptive construction heuristics, but local search is omitted.

Cano, J., R. Cabrera, and J. Vega (2002). Procedimientos constructivos adaptivos (GRASP) para el problema del empaquetado bidimensional. *Revista Iberoamericana de Inteligencia Artificial* **15**, 26–33.

To solve the bidimensional packing problem, several constructive adaptive heuristics are proposed. Some of them only have a GRASP construction phase, while others apply also a local search phase. Computational results show that in many cases the proposed heuristics obtain the optimal solution. In Spanish.

Cano, J., O. Cordón, F. Herrera, and L. Sánchez (2002a). A greedy randomized adaptive search procedure applied to the clustering problem as an initialization process using K-means as a local search procedure. *Journal of Intelligent and Fuzzy Systems* **12**, 235–242.

A GRASP is proposed for cluster analysis using a probabilistic greedy Kaufman initialization in the construction phase and K-Means as local search procedure.

Cano, J., O. Cordón, F. Herrera, and L. Sánchez (2002b). A GRASP algorithm for clustering. In F. J. Garijo, J. Santos, and M. Toro (Eds.), *Advances in Artificial Intelligence - IBERAMIA 2002, 8th Ibero-American Conference on AI, Seville, Spain, November 12-15, 2002, Proceedings*, Volume 2527 of *Lecture Notes in Computer Science*, pp. 214–223. Springer.

A GRASP for cluster analysis is described. See also page 362 and page 388.

Catalano, M. F. and F. Malucelli (2000). Parallel randomized heuristics for the set covering problem. Technical report, Transportation and traffic engineering section, Delft U. of Technology, 2600 AG Delft, The Netherlands. To appear in International J. of Computer Research.

A general scheme to design heuristics for the set covering problem is proposed. A first group of procedures randomize the choice of the next element to be added at the solution under construction in a way similar to ant system, while a second set of procedures introduces a random perturbation of the costs of the problem instance. The second set includes also a GRASP.

Chardaire, P., G. McKeown, and J. Maki (2001). Application of GRASP to the multiconstraint knapsack problem. In E. B. et al. (Ed.), *EvoWorkshop 2001*, pp. 30–39. Springer-Verlag Berlin Heidelberg.

Several implementations of GRASP for the multiconstraint knapsack problem are presented. In all implementations, the greedy functions are based on the profit per weight unit associated with each element. 1-opt and 2-opt search strategies are used in the local search phase.

Delorme, X., X. Gandibleux, and J. Rodriguez (2001). GRASP for set packing problems. In *Proceedings of the Operational Research Peripatetic Post-Graduate Programme (ORP3)*.

Two GRASP implementations for the set packing problem are proposed. The first GRASP is inspired by a GRASP for the set covering problem that appeared in the literature. The neighborhood structure adopted is a *k-p* exchange, which consists in fixing to 0 the value of *k* variables and to 1 the value of the remaining *p* variables. The 0-1, 1-1, 2-1, and 1-2 exchange neighborhoods are investigated. The second GRASP is inspired by a GRASP for the node packing problem that appeared in the literature and uses a 1-2 exchange neighborhood.

Delorme, X., X. Gandibleux, and J. Rodriguez (2004). GRASP for set packing problems. *European J. of Operational Research* **153**(3), 564–580.

GRASP is applied to solve the set packing problem. Several construction phases are studied and improvements based on advanced strategies are evaluated. These include reactive GRASP, path relinking, and a procedure involving the diversification of the selection (using a learning process).

Feo, T. and M. Resende (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters* **8**, 67–71.

GRASP is proposed for a class of difficult set covering problems that arise in computing the 1-width of the incidence matrix of Steiner triple systems. A value based restricted candidate list is used in the construction phase. The local search is based on the elimination of redundant elements in the cover. See also page 351.

Hammer, P. and D. Rader, Jr. (2001). Maximally disjoint solutions of the set covering problem. *Journal of Heuristics* **7**, 131–144.

The problem of finding two solutions of a set covering problem that have a minimum number of common variables is addressed. It is proved that this problem is NP-complete and three heuristics are proposed for solving it. Two of these algorithms find the solutions sequentially. One of them is a GRASP.

Klincewicz, J. and A. Rajan (1994). Using GRASP to solve the component grouping problem. *Naval Research Logistics* **41**, 893–912.

Two new heuristics are proposed for solving a particular set partitioning problem that arises in robotics assembly, as well as in a number of other manufacturing and material logistics application areas. The heuristics are GRASPs involving two alternate procedures for determining starting i points: component-based and code-based. See also page 410.

Laguna, M., T. Feo, and H. Elrod (1994). A greedy randomized adaptive search procedure for the two-partition problem. *Operations Research* **42**, 677–687.

A GRASP for the network 2-partition problem is proposed. The greedy function of the construction phase minimizes the augmented weight of the partition. For the local improvement phase, four alternative procedures are considered: best swap, first swap, slight swap, and slightest swap. The best strategies are slight and slightest swaps. Slight swap selects a near-minimum gain exchange at each iteration, while slightest swap chooses the absolute minimum gain.

Pacheco, J. and O. Valencia (2003). Design of hybrids for the minimum sum-of-squares clustering problem. *Computational Statistics and Data Analysis* **43**, 235–248.

Several heuristics are proposed for the non-hierarchical clustering problem under the criterion of minimum sum-of-squares clustering. These heuristics incorporate genetic operators, local search, and tabu search. They are compared with other heuristic approaches, including a GRASP, on a set of test problems. See also page 364.

Resende, M. (1998). Computing approximate solutions of the maximum covering problem using GRASP. *Journal of Heuristics* **4**, 161–171.

A GRASP for the maximum covering problem is described. For details about the GRASP implementation, see page 419.

13.11 ROUTING

Routing problems arise in transportation, telecommunications, and waste management. Such applications are contained in the papers below.

Bibliography

Arakaki, R. (1998). O problema de roteamento de veículos e algumas metaheurísticas. Master's thesis, Instituto Nacional de Pasquisas Espaciais, Brazil.

> Several metaheuristics are proposed for the vehicle routing problem, including a GRASP. In the GRASP construction phase, a distance function is used as the greedy function, while the local search tries to find a better allocation for a costumer at a time. In Portuguese.

Argüello, M., J. Bard, and G. Yu (1997). A GRASP for aircraft routing in response to groundings and delays. *Journal of Combinatorial Optimization* **1**, 211–228.

> A GRASP is presented to reconstruct aircraft routings in response to groundings and delays experienced over the course of the day. The objective is to minimize the cost of reassigning aircraft to flights taking into account available resources and other system constraints. See also page 411.

Baker, B. and C. Carreto (2003). A visual interactive approach to vehicle routing. *Computers and Operations Research* **30**, 321–337.

> A graphical-user-interface (with a Microsoft Windows interface) and a GRASP based heuristic for the basic vehicle routing problem are described. The proposed visual interactive system, CRUISE, provides high flexibility. Although the best known results on VRP benchmarks are obtained by tabu search and simulated annealing algorithms, none of them allows the user any control for combining their insights and knowledge. See also page 411.

Bard, J., L. Huang, P. Jaillet, and M. Dror (1998). A decomposition approach to the inventory routing problem with satellite facilities. *Transportation Science* **32**, 189–203.

> A methodology is presented that decomposes the inventory routing problem with satellite facilities over the planning horizon, and then solves daily rather than multi-day vehicle routing problems. Three heuristics are proposed for solving the vehicle routing problem with satellite facilities: randomized Clarke-Wright, GRASP, and modified sweep. See also page 411.

Bard, J., G. Kontoravdis, and G. Yu (2002). A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science* **36**, 250–269.

A GRASP is proposed to obtain feasible solutions and/or upper bounds used in a branch-and-cut algorithm for the vehicle routing problem with time windows. See also page 412.

Carreto, C. and B. Baker (2002). A GRASP interactive approach to the vehicle routing problem with backhauls. In C. Ribeiro and P. Hansen (Eds.), *Essays and surveys in metaheuristics*, pp. 185–200. Kluwer Academic Publishers.

A GRASP for the vehicle routing problem with backhauls is proposed. The construction phase is implemented in a clustering heuristic that constructs the routes by clustering the remaining customers according to the vehicles defined by seeds while applying the 3-opt heuristic to reduce the total distance traveled by each vehicle. The greedy function takes into account routes with smallest insertion cost and costumers with biggest difference between the smallest and the second smallest insertion costs and smallest number of routes they can traverse. As the local search phase, 3-opt is used. See also page 363.

Chaovalitwongse, W., D. Kim, and P. Pardalos (2003). GRASP with a new local search scheme for vehicle routing problems with time windows. *J. of Combinatorial Optimization* **7**, 179–207.

A GRASP is proposed for minimizing the number of needed vehicles and the travel distances in the vehicle routing problem with time windows. The search method proposed uses a combination of random and greedy functions.

Corberán, A., R. Martí, and J. Sanchís (2002). A GRASP heuristic for the mixed Chinese postman problem. *European Journal of Operational Research* **142**, 70–80.

The construction phase of the GRASP proposed to solve the mixed Chinese postman problem uses a greedy function based on the definition of the degree of a node in terms of both incident oriented arcs and incident undirected edges. Each iteration of the local search procedure selects a pair of vertices $u, v \in V$ that are candidates for the move if they are joined by a path of duplications. See also page 374.

Hjorring, C. (1995). *The vehicle routing problem and local search metaheuristics*. Ph. D. thesis, University of Auckland, Auckland, New Zealand.

Three metaheuristics for effectively searching through the space of cyclic orders are developed. They are based on GRASP, tabu search, and genetic algorithms. For tabu search, different schemes are investigated to control the tabu list length, including a reactive tabu search method. To obtain good solutions when using the genetic algorithm, specialized crossovers are developed, and a local search component is added. GRASP is used to construct an initial good solution.

Kontoravdis, G. and J. Bard (1995). A GRASP for the vehicle routing problem with time windows. *ORSA Journal on Computing* **7**, 10–23.

A GRASP is proposed for minimizing the fleet size of temporarily constrained vehicle routing problems with two types of service. The greedy function of the construction phase takes into account both the overall minimum insertion cost and the penalty cost. Local search is applied to the best solution found every five iterations of the first phase, rather than to each feasible solution.

Marinakis, Y. and A. Migdalas (2003). Expanding neighborhood GRASP for the Traveling Salesman Problem. Technical report, Technical University of Crete, Chania 73100, Greece.

A GRASP that includes tour improvement methods is proposed for the traveling salesman problem. See also page 363 and page 376.

Resende, L. and M. Resende (1999). A GRASP for frame relay PVC routing. In *Proc. of the Third Metaheuristics International Conference*, pp. 397–402.

A GRASP is described for routing permanent virtual circuits (PVC) for frame relay in telecommunications systems. The objective is to minimize PVC delays while balancing trunk loads. The greedy choice selects from the set of not yet routed PVCs the one that minimizes the delay while balancing the trunk loads. See also page 419.

Resende, M. and C. Ribeiro (2003). GRASP with path-relinking for private virtual circuit routing. *Networks* **41**, 104–114.

A frame relay service offers virtual private networks to customers by provisioning a set of long-term private virtual circuits (PVCs) between customer endpoints on a large backbone network. During the provisioning of a PVC, routing decisions are made without any knowledge of future requests. Over time, these decisions can cause inefficiencies in the network and occasional offline rerouting of the PVCs is needed. The offline PVC routing problem is formulated as an integer multicommodity flow problem with additional constraints and with an objective function that minimizes delays and network overload. Variants of a GRASP with path-relinking heuristic are proposed for this problem. See also page 364 and page 419.

13.12 SEQUENCING AND SCHEDULING

GRASP has been applied to numerous sequencing and scheduling problems. The papers in this section illustrate this.

Bibliography

Aiex, R. (2002). *Uma investigação experimental da distribuição de probabilidade de tempo de solução em heurísticas GRASP e sua aplicação na análise de implementações paralelas*. Ph. D. thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil.

> A new methodology is described for the analysis of GRASP. Hybrid strategies with path-relinking are also proposed. These are studied on the 3-index assignment problem as well as the job shop scheduling problem. See also page 369 and page 381.

Aiex, R., S. Binato, and M. Resende (2003). Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing* **29**, 393–430.

> A parallel GRASP with path-relinking as an intensification strategy for the job shop problem is described based on some ideas proposed in the GRASP of Binato et al. (2002). See also page 361 and page 369.

Akturk, M. and K. Kiliç (1999). Generating short-term observations for space mission projects. *J. of Intelligent Manufacturing* **10**, 387–404.

> Generating short-term observations for space mission projects is basically a scheduling problem. It consists in generating short-term observation schedules of Hubble Space Telescope (HST) such that the scientific return is maximized. A new dispatching rule and a set of local search based algorithms (including a GRASP) are proposed.

Akturk, M. and D. Ozdemir (2001). A new dominance rule to minimize total weighted tardiness with unequal release dates. *European Journal of Operational Research* **135**, 394–412.

> A new algorithm that provides sufficient condition for local optimality and that can be embedded into several heuristic frameworks (including GRASP) is proposed.

Atkinson, J. (1998). A greedy randomised search heuristic for time-constrained vehicle scheduling and the incorporation of a learning strategy. *Journal of the Operational Research Society* **49**, 700–708.

Two forms of adaptive search called *local* and *global* adaptation are identified. In both search techniques, the greedy function takes into account a quantity that measures heuristically the quality of the partial solution. While in local adaptation the decisions made within a particular run influence only the subsequent performance of the heuristic, global adaptation involves making decisions that affect the performance of the heuristic in subsequent runs. See also page 411.

Bard, J. and T. Feo (1989). Operations sequencing in discrete parts manufacturing. *Management Science* **35**, 249–255.

A method for efficiently sequencing cutting operations associated with the manufacture of discrete parts is proposed. The problem is modeled as an integer program. This is relaxed via Lagrangian relaxation into a min-cut problem on a bipartite network. To obtain lower bounds, a max-flow algorithm is applied and the corresponding solution is input to a GRASP. See also page 409.

Bard, J., T. Feo, and S. Holland (1996). A GRASP for scheduling printed wiring board assembly. *I.I.E. Transactions* **28**, 155–165.

The assembly of printed wiring boards (PWBs) typically involves the coordination of thousands of components and hundreds of part numbers in a job shop environment with more than 50 different processes and workstations. A GRASP is proposed for solving the daily scheduling problem that arises in such environment. See also page 409.

Binato, S., W. Hery, D. Loewenstern, and M. Resende (2002). A greedy randomized adaptive search procedure for job shop scheduling. In C. Ribeiro and P. Hansen (Eds.), *Essays and surveys in metaheuristics*, pp. 58–79. Kluwer Academic Publishers.

A GRASP is designed, incorporating an intensification strategy and a POP (Proximate Optimality Principle) in the construction phase. The greedy criterion is to minimize the makespan resulting from the addition of an operation to the schedule under construction, while the local search procedure uses a 2-exchange neighborhood.

Casey, S. and J. Thompson (2003). GRASPing the examination scheduling problem. In E. Burke and P. D. Causmaecker (Eds.), *PATAT 2002*, Volume 2740 of *Lecture Notes in Computer Science*, pp. 232–244. Springer.

Reactive GRASP is applied to the examination scheduling problem. A comparison with other algorithms for this problem shows that GRASP is a powerful solution method.

Christofoletti, L. (2002). Métodos de reinício aplicados ao seqüenciamento em uma máquina com tempos de preparação e data de entrega. Master's thesis, Departamento de Engenharia de Sistemas, Universidade Estadual de Campinas, Brazil.

This thesis considers applications of GRASP and its variants to the problem of minimizing the total tardiness of jobs on a single machine with sequence dependent setup times. An innovative aspect of this thesis is the introduction in the multistart framework with some memory strategies for storing a population of high quality solutions. In Portuguese.

Christofoletti, L. and V. Armentano (2001). Estratégias de reinício de busca local baseadas em memória para programação de tarefas em uma máquina. In *Proceedings of the XXXIII Brazilian Symposium on Operations Research*, pp. 1381–1389.

It is shown that incorporating adaptive memory in multistart random methods improves their performance. To validate their thesis, the paper addresses the problem of minimizing total job tardiness on a single machine with sequence dependent setup times. A GRASP is implemented, combined with some basic principles of memory utilization during the construction phase. In Portuguese.

De, P., J. Ghosj, and C. Wells (1994). Solving a generalized model for CON due date assignment and sequencing. *International Journal of Production Economics* **34**, 179–185.

A generalized model for assigning a constant flow allowance (CON) due date to a set of jobs and sequencing them on a single machine is considered. The problem is viewed as a 0-1 quadratic problem and a GRASP is proposed to solve the quadratic problem. The randomization strategy used is inspired by a gradient-based variable forcing methodology proposed by Pardalos and Rodgers (1990) for a branch & bound algorithm. The local search procedure is based on a definition of neighborhood in which two solutions are neighbors if they differ in the value of exactly one variable.

Feo, T. and J. Bard (1989). Flight scheduling and maintenance base planning. *Management Science* **35**, 1415–1432.

A model is presented that can be used by planners to both locate maintenance stations and develop flight schedules that better meet the cyclical demand for maintenance. See also page 412.

Feo, T., J. Bard, and S. Holland (1995). Facility-wide planning and scheduling of printed wiring board assembly. *Operations Research* **43**, 219–230.

A decision support system known as INSITES is described. INSITES was designed to assist Texas Instruments in the day-to-day assembly operations of their printed wiring board (PWB) facilities. A GRASP is used to solve the underlying multiple machine scheduling problem. See page 410 for details of the GRASP implementation.

Feo, T., K. Sarathy, and J. McGahan (1996). A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. *Computers & Operations Research* **23**, 881–895.

A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties is presented. The greedy function of the GRASP construction phase proposed is made up of two components: the switch over cost and the opportunity cost associated with not inserting a specific job in the next position and instead, inserting it after half of the unscheduled jobs have been scheduled. This greedy function tends to lead to a balance between the natural order and nearest neighbor approaches. The local search uses 2-exchange, insertion exchange, and a combination of the two.

Feo, T., K. Venkatraman, and J. Bard (1991). A GRASP for a difficult single machine scheduling problem. *Computers & Operations Research* **18**, 635–643.

GRASP is applied to an unusually difficult scheduling problem with flow time and earliness penalties. Two greedy functions are developed and tested. The first is the difference between the flow time and earliness penalties, normalized by the processing time. The second function evaluates the cost of scheduling a job next by estimating the cost of the remaining schedule. The local search uses 2-exchange and insertion exchange.

García, J., S. Lozano, K. Smith, and F. Guerrero (2001). A comparison of GRASP and an exact method for solving a production and delivery scheduling problem. In *First International Workshop on Hybrid Intelligent Systems (HIS'01), Adelaide, Australia*.

An exact approach and a GRASP are proposed to solve a production and delivery scheduling problem. The greedy criterion takes into account order weights, while the local search procedure uses an exchange neighborhood. See also page 413.

Laguna, M. and J. González-Velarde (1991). A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent Manufacturing* **2**, 253–260.

A hybrid GRASP/tabu search metaheuristic is proposed for the weighted earliness penalty problem with deadlines in identical parallel machines.

Lourenço, H., J. Paixão, and R. Portugal (2001). Multiobjective metaheuristics for the bus-driver scheduling problem. *Transportation Sciences* **35**, 331–343.

Several metaheuristics are presented to solve real driver scheduling problems in public transportation bus companies. They include a GRASP. See also page 413.

Ríos-Mercado, R. and J. Bard (1998). Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, 76–98.

Two new heuristics are presented for the flowshop scheduling problem with sequence-dependent setup times and makespan minimization objective, one of which is a GRASP.

Rivera, L. (1998). Evaluation of parallel implementations of heuristics for the course scheduling problem. Master's thesis, Instituto Tecnologico y de Estudios Superiores de Monterrey, Monterrey, Mexico.

Several parallel implementations of heuristics are proposed for the course scheduling problem, including a GRASP. In Spanish. See also page 372.

Rojanasoonthon, S. and J. Bard (2004). A GRASP for parallel machine scheduling with time windows. *INFORMS Journal on Computing*. To appear.

A GRASP is designed for a parallel machine scheduling problem with time windows. This extends the GRASP proposed in Rojanasoonthon, Bard, and Reddy (2003) to solve parallel machine scheduling problems in the presence of time windows.

Rojanasoonthon, S., J. Bard, and S. Reddy (2003). Algorithms for parallel machine scheduling: A case study of the tracking and data relay satellite system. *Journal of the Operational Research Society* **54**, 806–821.

A GRASP is designed for a parallel machine scheduling problem. The greedy criterion in the construction phase is based on maximizing a flexibility function that measures how much slack a schedule has after the insertion is made.

Souza, M., N. Maculan, and L. Ochi (2003). A GRASP-tabu search algorithm for school timetabling problems. In M. Resende and J. de Sousa (Eds.), *Metaheuristics: Computer decision-making*, pp. 659–672. Kluwer Academic Publishers.

A hybrid approach is proposed for school timetabling problems. It uses a greedy randomized construction phase for obtaining a feasible solution to be possibly improved applying a tabu search. The greedy choice first takes into account the number of available teachers and then the activity degree of each teacher. A timetable is represented as a $m \times q$ matrix Q of integer values, such that each row i represents the weekly schedule of teacher i and q_{ik} represents the activity of teacher i in period k. A neighbor of a timetable Q is a timetable Q', obtained from Q simply by changing two different and nonnegative values of a give row of Q. See also page 365.

Xu, J. and S. Chiu (1996). Solving a real-world field technician scheduling problem. In *Proceedings of the International Conference on Management Science and the Economic Development of China*, pp. 240–248.

The objective of the field technician scheduling problem is to assign a set of jobs at different locations with time windows to technicians with different job skills. The greedy choice of the proposed GRASP is to select jobs with the highest unit weight. See also page 420.

Xu, J. and S. Chiu (2001). Effective heuristic procedure for a field technician scheduling problem. *Journal of Heuristics* **7**, 495–509.

Several heuristics, including a GRASP, are designed and tested for solving the field technician scheduling problem. See also page 420.

13.13 LOGIC

GRASP has been applied to problems in logic, including SAT, MAX-SAT, and logical clause inference, as shown by the following papers.

de Campos, L., J. Fernández-Luna, and J. Puerta (2002). Local search methods for learning Bayesian networks using a modified neighborhood in the space of DAGs. In F. J. Garijo, J. Riquelme, and M. Toro (Eds.), *Advances in Artificial Intelligence - IBERAMIA 2002, 8th Ibero-American Conference on AI, Seville, Spain, November 12-15, 2002, Proceedings*, Volume 2527 of *Lecture Notes in Computer Science*, pp. 182–192. Springer.

GRASP is applied to learning Bayesian networks from data. The GRASP construction phase is a randomization of the greedy algorithm of Butine (1991) and the local search is a hill-climbing algorithm in the space of directed acyclic graphs (DAGs). The GRASP obtains excellent results in the computational experiments described.

Deshpande, A. and E. Triantaphyllou (1998). A greedy randomized adaptive search procedure (GRASP) for inferring logical clauses from examples in polynomial time and some extensions. *Mathematical and Computer Modelling* **27**, 75–99.

Two heuristics (one of which is a GRASP) are presented for inferring a small size Boolean function from complete and incomplete examples in polynomial time. Each example can be positive or negative depending on whether it must be accepted or rejected, respectively, by the target function. Both of the proposed heuristics are randomized in the sense that instead of choosing the best candidate element, a candidate list is built whose elements are assigned with evaluative function values close to the highest one.

Pardalos, P., L. Pitsoulis, and M. Resende (1996). A parallel GRASP for MAX-SAT problems. *Lecture Notes in Computer Science* **1184**, 575–585.

A parallel GRASP for weighted maximum satisfiability (MAX-SAT) problem is proposed. The GRASP is based on the serial GRASP presented by Resende, Pitsoulis, and Pardalos (1997). See also page 372.

Resende, M. and T. Feo (1996). A GRASP for satisfiability. In D. Johnson and M. Trick (Eds.), *Cliques, Coloring, and Satisfiability: The Second DIMACS Implementation Challenge*, Volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 499–520. American Mathematical Society.

A GRASP is described for the satisfiability problem. It can be also directly applied to both the weighted and unweighted versions of the maximum satisfiability problem. The adaptive greedy function is a hybrid combination of two functions. One function seeks to maximize the number of yet-unsatisfied clauses that become satisfied after the assignment of each construction iteration, while the other maximizes the number of yet-unassigned literals in yet-unsatisfied clauses that become satisfied if opposite assignments were to be made. The local search flips the assignment of each variable, one at a time, checking if the new truth assignment increases the number of satisfied clauses.

Resende, M., L. Pitsoulis, and P. Pardalos (1997). Approximate solution of weighted MAX-SAT problems using GRASP. In J. Gu and P. Pardalos (Eds.), *Satisfiability problems*, Volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 393–405. American Mathematical Society.

A GRASP is proposed for finding approximate solutions of weighted MAX-SAT problems. The greedy adaptive function is to maximize the total weight of yet-unsatisfied clauses that become satisfied after the assignment of each construction phase iteration. The local search uses the 1-flip neighborhood of a vector x, defined as the set of all binary vectors that differ from x in exactly one literal.

Resende, M., L. Pitsoulis, and P. Pardalos (2000). Fortran subroutines for computing approximate solutions of MAX-SAT problems using GRASP. *Discrete Applied Mathematics* **100**, 95–113.

A set of Fortran subroutines for computing approximate solutions of MAX-SAT problems is described. The algorithm implemented was proposed by Resende, Pitsoulis, and Pardalos (1997). Two versions of the subroutines are distributed. One version uses a neighborhood data structure in order to speed up the local search phase, while the second version, since it does not make use of this data structure, is more memory efficient but less time efficient. Computational results improve upon those in Resende, Pitsoulis, and Pardalos (1997) using an RCL parameter α randomly chosen each GRASP iteration from the interval $[0, 1]$. See also page 368.

Yilmaz, E., E. Triantaphyllou, J. Chen, and T. Liao (2003). A heuristic for mining association rules in polynomial time. *Mathematical and Computer Modelling* **37**, 219–233.

The problem consisting in mining association rules in a database needs to be efficiently solved, especially nowadays when modern databases have very large sizes. The authors propose a heuristic algorithm that incorporates the randomized idea of the GRASP construction phase.

13.14 MANUFACTURING

GRASP has been used to address several applications in manufacturing. The following papers are examples of this.

Bibliography

Bard, J. and T. Feo (1989). Operations sequencing in discrete parts manufacturing. *Management Science* **35**, 249–255.

An integer program is relaxed via Lagrangian relaxation into a min-cut problem on a bipartite network. To obtain lower bounds, a max-flow algorithm is applied and the corresponding solution is input to a GRASP. See also page 402.

Bard, J. and T. Feo (1991). An algorithm for the manufacturing equipment selection problem. *IIE Transactions* **23**, 83–92.

The objective is to determine how many of each machine type to purchase and what fraction of the time each piece of equipment will be configured for a particular type of operation. The problem is converted into a MILP and a depth-first branch & bound algorithm is used, employing the greedy randomized set covering heuristic of Feo and Resende (1989), to implicitly search for optimality.

Bard, J., T. Feo, and S. Holland (1996). A GRASP for scheduling printed wiring board assembly. *I.I.E. Transactions* **28**, 155–165.

A GRASP is proposed for solving the daily scheduling problem that arises in a job shop environment with more than 50 different processes and workstation. See also page 402.

Bautista, J., R. Suárez, M. Mateo, and R. Companys (2000). Local search heuristics for the assembly line balancing problem with incompatibilities between tasks. In *Proceedings of the IEEE ICRA-00*, Volume 3, pp. 2404–2409.

The assembly line balancing problem with incompatibilities between tasks consists in minimizing the total number of needed workstations and minimizing the the cycle time for the minimum number of workstations. A GRASP and a genetic algorithm are proposed for solving the problem. The greedy choice favors tasks with the best index value, while the local search phase simply changes the order of elements in the sequence solution.

Feo, T. and J. Bard (1989). The cutting path and tool selection problem in computer-aided process planning. *Journal of Manufacturing Systems* **8**, 17–26.

A method for minimizing the sum of tool setup and volume removal times associated with metal cutting operations on a flexible machine is given. The problem

is modeled as an integer program, then relaxed into a min-cut problem on a simple network. After obtaining a tentative solution, a GRASP is used to identify good feasible points corresponding to alternative process plans. These are seen to speed convergence during branch & bound.

Feo, T., J. Bard, and S. Holland (1995). Facility-wide planning and scheduling of printed wiring board assembly. *Operations Research* **43**, 219–230.

A GRASP is used to solve a multiple machine scheduling problem. The schedule produced at each GRASP iteration is evaluated based on one of five different optimization criteria. The choice of the criterion to be followed is made by the user to rank order the schedules provided by multiple GRASP iterations. See page 403.

Klincewicz, J. and A. Rajan (1994). Using GRASP to solve the component grouping problem. *Naval Research Logistics* **41**, 893–912.

Two new GRASPs are proposed that involve two alternate procedures for determining starting points: component-based and code-based. See also page 395.

Yen, J., M. Carlsson, M. Chang, J. Garcia, and H. Nguyen (2000). Constraint solving for inkjet print mask design. *Journal of Imaging Science and Technology* **44**, 391–397.

Print masks are used to determine which nozzles on an inkjet printer cartridge are to spit an ink droplet at each particular instant in a multiple-pass print mode. A GRASP is proposed for for automatic generation of print masks and has been used to design the print masks for Hewlett Packard's wide format printers (DeskJet 2500C and 2500CM).

13.15 TRANSPORTATION

GRASP has been used to find approximate solutions of problems in air, rail, and intermodal transportation. The following papers illustrate these applications.

Bibliography

Argüello, M., J. Bard, and G. Yu (1997). A GRASP for aircraft routing in response to groundings and delays. *Journal of Combinatorial Optimization* **1**, 211–228.

A neighborhood search technique is proposed that takes as input an initial feasible solution, so that the construction phase is omitted. Two types of partial route exchange operations are described. The first exchanges flight sequences with identical endpoints and in the second sequence of flights being exchanged must have the same origination airport, but the termination airports are swapped. See also page 397.

Atkinson, J. (1998). A greedy randomised search heuristic for time-constrained vehicle scheduling and the incorporation of a learning strategy. *Journal of the Operational Research Society* **49**, 700–708.

In both adaptive search techniques proposed and called *local* and *global* adaptations, respectively, the greedy function takes into account a quantity that measures heuristically the quality of the partial solution. While in local adaptation the decisions made within a particular run influence only the subsequent performance of the heuristic, global adaptation involves making decisions that affect the performance of the heuristic in subsequent runs. See also page 402.

Baker, B. and C. Carreto (2003). A visual interactive approach to vehicle routing. *Computers and Operations Research* **30**, 321–337.

See page 397.

Bard, J. (1997). An analysis of a rail car unloading area for a consumer products manufacturer. *Journal of the Operational Research Society* **48**, 873–883.

Discussion of design and analysis of the railcar unloading area of Proctor & Gamble's principal laundry detergent plant. To solve the problem, four alternatives are proposed and evaluated with the help of a GRASP. See also page 381.

Bard, J., L. Huang, P. Jaillet, and M. Dror (1998). A decomposition approach to the inventory routing problem with satellite facilities. *Transportation Science* **32**, 189–203.

Three heuristics are proposed for solving the vehicle routing problem with satellite facilities: randomized Clarke-Wright, GRASP, and modified sweep. The

GRASP proposed is a modified version of the GRASP of Kontoravdis and Bard (1995). See also page 397.

Bard, J., G. Kontoravdis, and G. Yu (2002). A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science* **36**, 250–269.

A GRASP is proposed to obtain feasible solutions and/or upper bounds used in a branch-and-cut algorithm for the vehicle routing problem with time windows. See also page 398.

Campbell, A. and M. Savelsbergh (2002). Decision support for consumer direct grocery initiatives. Technical report, Department of Management Sciences, Tippie College of Business, University of Iowa.

Business-to-Consumer e-commerce has led to the proposal of new consumer direct service models and activities, such as grocery delivery services. The seller has to decide which request to accept and for each accepted request he has to establish the time slot when the delivery is going to be done. Insertion based heuristics are proposed. To improve the chances that a delivery request can be taken, randomization is used as in the GRASP proposed by Kontoravdis and Bard (1995).

Campbell, A. and M. Savelsbergh (2003). Incentive schemes for consumer direct delivery. Technical report, Department of Management Sciences, Tippie College of Business, University of Iowa.

A different aspect of a problem arising in Business-to-Consumer e-commerce is addressed, i.e the promise of a delivery window. Several approaches are proposed, including a GRASP. The greedy criterion is based on the costs of inserting a delivery into a feasible schedule.

Delorme, X., J. Rodriguez, and X. Gandibleux (2001). Heuristics for railway infrastructure saturation. In C. Zaroliagis (Ed.), *Electronic Notes in Theoretical Computer Science*, Volume 50. Elsevier.

To evaluate railway infrastructure capacity, two heuristics approaches are proposed, including a GRASP. The greedy function is defined on the number of mathematical model constraints concerned by decision variables, while local search procedure uses a k-p exchange neighborhood.

Feo, T. and J. Bard (1989). Flight scheduling and maintenance base planning. *Management Science* **35**, 1415–1432.

The problem is formulated as a minimum cost multicommodity flow network with integral constraints, where each airplane represents a separate commodity and each arc has an upper and lower capacity of flow. Since obtaining feasible solutions from the LP relaxation is difficult, the authors propose a GRASP. See page 403.

Feo, T. and J. González-Velarde (1995). The intermodal trailer assignment problem: Models, algorithms, and heuristics. *Transportation Science* **29**, 330–341.

The problem is formulated as a set covering problem. A branch-and-bound algorithm and a GRASP are developed for solving it. The greedy strategy of the construction phase of GRASP consists in selecting at each step a feasible assignment of the most difficult to use available railcar together with the most difficult

to assign trailer. To improve the constructed solution, a 2-exchange local search is applied, carrying out a complete enumeration of the solutions in the neighborhood. See also page 382.

García, J., S. Lozano, K. Smith, and F. Guerrero (2001). A comparison of GRASP and an exact method for solving a production and delivery scheduling problem. In *First International Workshop on Hybrid Intelligent Systems (HIS'01), Adelaide, Australia*.

See also page 404.

Lourenço, H., J. Paixão, and R. Portugal (2001). Multiobjective metaheuristics for the bus-driver scheduling problem. *Transportation Sciences* **35**, 331–343.

To design GRASP, a set N of n duties is defined and a greedy criterion based on a quantity proportional to the cost associated with the duties is used. The local search procedure uses a 1-exchange neighborhood. See also page 404.

Milidíu, R., A. Pessoa, V. Braconi, E. Laber, and P. Rey (2001). Um algoritmo GRASP para o problema de transporte de derivados de petróleo em oleodutos. In *Proceedings of the XXXIII Brazilian Symposium on Operations Research*, pp. 237–246.

A GRASP is described for petroleum derivatives transportation in pipelines. The greedy function is a simple cost function obtained as sum of the total volumes of the product.

Sosnowska, D. (2000). Optimization of a simplified fleet assignment problem with metaheuristics: Simulated annealing and GRASP. In P. Pardalos (Ed.), *Approximation and complexity in numerical optimization*. Kluwer Academic Publishers.

A simulated annealing and a GRASP are proposed. In GRASP only exchanges leading to a better solution are permitted and the potentially best part of the assignment is conserved and the rest is randomly reattributed. The construction phase does not use a restricted candidate list explicitly, but a solution is built by simply trying to make the time interval between two flights as small as possible. See also page 386.

13.16 TELECOMMUNICATIONS

GRASP has been widely applied in the telecommunications field to problems ranging from network design to facility location and routing. Below are papers describing applications of GRASP in telecommunications.

Abello, J., P. Pardalos, and M. Resende (1999). On maximum clique problems in very large graphs. In J. Abello and J. Vitter (Eds.), *External memory algorithms and visualization*, Volume 50 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 119–130. American Mathematical Society.

A semi-external memory GRASP is presented to approximately solve the maximum clique problem and maximum quasi-clique problem in very large graphs. Communities of interest are extracted from telephone call detail graphs. See also page 373.

Abello, J., M. Resende, and S. Sudarsky (2002). Massive quasi-clique detection. *Lecture Notes in Computer Science* **2286**, 598–612.

A GRASP is designed for extracting dense subgraphs on case of both nonbipartite and bipartite case. Communities of interest are extracted from telephone call detail graphs. See also page 373.

Álvarez, A., J. L. González, and K. De-Alba (2003). Scatter search for a network design problem. Technical Report PISIS-2003-02, Universidad Autónoma de Nuevo León, Facultad de Ingeniería Mecánica y Eléctrica, División de Posgrado en Ingeniería de Sistemas, Mexico.

The authors propose a Scatter Search (SS) algorithm for solving a fixed charge capacitated multicommodity network design problems on undirected networks. A GRASP-based diversification generation method (DGM) with memory features is described. In the GRASP DGM, for each commodity a certain number q of shortest paths between each origin-destination pair are kept as RCL elements. The local search consists basically in sorting the chosen paths and possibly exchange some of them in order to get a better distribution. See also page 362.

Amaldi, E., A. Capone, and F. Malucelli (2003). Planning UMTS base station location: Optimization models with power control and algorithms. *IEEE Transactions on Wireless Communications* **2**(5), 939–952.

The greedy function takes into account the fraction of traffic covered and the installation costs. Local search is a swap procedure. See also page 387.

Amaldi, E., A. Capone, F. Malucelli, and F. Signori (2003). Optimization models and algorithms for downlink UMTS radio planning. In *Wireless Communications and Networking, 2003 (WCNC 2003)*, Volume 2, pp. 827–831.

The UMTS base station location problem is addressed. Previously, in Amaldi et al. (2003), two randomized heuristics were proposed. They are here adapted for solving a similar problem. See also page 388.

Armony, M., J. Klincewicz, H. Luss, and M. Rosenwein (2000). Design of stacked self-healing rings using a genetic algorithm. *Journal of Heuristics* **6**, 85–105.

A genetic algorithm for design of stacked self-healing rings is proposed. The objective is to optimize the trade-off between the cost of connecting nodes to the ring and the cost of routing demand on multiple rings. The initial population of the genetic algorithm is made up of randomly generated solutions as well as solutions generated by a GRASP. Computational comparisons are made with a commercial integer programming package.

Brunato, M. and R. Battiti (2001). A multistart randomized greedy algorithm for traffic grooming on mesh logical topologies. Technical report, Department of Mathematics, University of Trento, Trento, Italy.

A logical topology design problem on Dense Wavelength Division Multiplexing (DWDM) optical networks is addressed. Traffic is measured at sub-wavelength resolution and the key factor to determine the fitness of a solution is the number of lightpaths required. A GRASP-like heuristic for minimizing the number of lightpaths is described. The greedy choice takes into account the load associated with each lightpath.

Canuto, S., M. Resende, and C. Ribeiro (2001). Local search with perturbation for the prize-collecting Steiner tree problems in graphs. *Networks* **38**, 50–58.

The prize collecting Steiner tree problem arises in telecommunications access network design. See page 357, page 362, and page 374.

Cooke, B., D. Kwon, D. Glotov, S. Schurr, D. Taylor, and T. Wittman (2002). Mobility management in cellular telephony. Technical report, Institute of Mathematics and its Applications, University of Minnesota, USA.

Four algorithms are proposed: a branch-and-bound, a metropolis algorithm with annealing, a genetic algorithm, and a greedy search. The latter has been derived from a GRASP proposed for the quadratic assignment problem.

Gendron, B., J.-Y. Potvin, and P. Soriano (2002). Diversification strategies in local search for a nonbifurcated network loading problem. *European Journal of Operational Research* **142**(2), 231–241.

A heuristic approach is proposed that alternates construction and local search phases. Initially, a construction method provides a feasible solution, while at subsequent construction steps, a diversification approach is adopted for exploiting information gathered along previous iterations. Two local searches are used: a pure descent search and a tabu search. An implementation of GRASP is proposed.

Klincewicz, J. (1992). Avoiding local optima in the *p*-hub location problem using tabu search and GRASP. *Annals of Operations Research* **40**, 283–302.

A GRASP is proposed for the p-hub location problem. Its local search procedure is based on a 2-exchange. See also page 390.

Klincewicz, J. (2002). Enumeration and search procedures for a hub location problem with economies of scale. *Annals of Operations Research* **110**, 107–122.

The greedy function of the GRASP takes into account the amount of originating and terminating traffic. The local search uses a $1-1$ swap neighborhood structure. See also page 390.

Kumaran, K., A. Srinivasan, Q. Wang, S. Lanning, and K. Ramakrishnan (2001). Efficient algorithms for location and sizing problems in network design. In *Global Telecommunications Conference, 2001 (GLOBECOM '01)*, Volume 4, pp. 2586–2590. IEEE.

The problems of location and sizing in network design are considered. Algorithms based on linear programming and a slightly modified GRASP are developed. In the GRASP, the construction phase is performed at random. See page 390 for details about the local search.

Li, B., F. Chen, and L. Yin (2000). Server replication and its placement for reliable multicast. In *Proceedings of the IEEE ICCCN-00*, pp. 396–401.

In a multicast network, packets are forwarded from a source (server) to group of receivers along a distribution tree, where the source is the root, the receivers are the leaves, and the multicast-capable routers are the internal nodes. The problem consists of placing multiple replicated servers within the multicast-capable routers. Several heuristics are proposed, including a GRASP. The greedy function is the router cost function, while the local search phase uses a k-exchange neighborhood structure with $k = 1$.

Liu, X., P. Pardalos, S. Rajasekaran, and M. Resende (2000). A GRASP for frequency assignment in mobile radio networks. In S. Rajasekaran, P. Pardalos, and F.Hsu (Eds.), *Mobile Networks and Computing*, Volume 52 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 195–201. American Mathematical Society.

A GRASP for frequency assignment is described. The construction phase uses two greedy functions. The first chooses a vertex from the set of unselected vertices with high saturation degrees. The second function is used to assign a frequency to the selected vertex. A frequency is selected from a set of permissible frequencies that contribute little additional cost to the objective function. See also page 382.

Mahey, P. and C. Ribeiro (2002). Modeling modern multimedia traffic. *Annals of Operations Research* **110**, 107–122.

Survey of state-of-the-art algorithms for solving optimal routing problems on multi-service communication problems.

Myslek, A. (2001). Greedy randomised adaptive search procedures (GRASP) for topological design of MPLS networks. In *Proceedings of the 8th Polish Teletraffic Symposium*.

The IP/MPLS network cost optimization problem of selecting localization of nodes and links, combined with link dimensioning is addressed. A GRASP is

proposed, whose greedy function uses demand flow allocation, while at each iteration of the local search phase, some selected nodes (or edges) become unavailable if provided and vice versa.

Myslek, A. and P. Karaś (2002). Heuristic methods for topological design of telecommunication networks. In *Proceedings of PGTS 2002*.

Given a list of potential node locations and a list of feasible interconnections between nodes, the generic topological network design problem consists in finding a network structure and a demand allocation pattern that minimizes the cost of the network. A pool of heuristics are proposed for solving the problem, including a Simulated Allocation (SAL) and a hybrid GRASP that uses SAL as local search.

Oliveira, C. and F. Gomes (1999). Two metaheuristics for channel allocation in mobile telephony. Technical report, Artificial Intelligence Laboratory, Universidade Federal do Ceará, Fortaleza, Brazil.

Two heuristics are proposed: GRASP and Asynchronous Team (A-Team). The construction phase of the proposed GRASP is realized by a procedure that at each step chooses the next antenna to which a frequency will be assigned. In the RCL construction, priority is given to transmitters with fewer options of frequency assignment. To implement the local search phase, a down hill algorithm is used. It performs random perturbations in the solution, exchanging the frequency of one antenna by another randomly chosen. See also age 383.

Pasiliao, E. (1998). A greedy randomized adaptive search procedure for the multicriteria radio link frequency assignment problem. Technical report, Department of ISE, University of Florida, Gainesville, FL 32611-6595.

The objective of the problem addressed here is to minimize the order and the span of the solution set. The local search procedure attempts to eliminate each channel from the communication network. See also page 384.

Poppe, F., M. Pickavet, P. Arijs, and P. Demeester (1997). Design techniques for SDH mesh-restorable networks. In *Proceedings of the European Conference on Networks and Optical Communications (NOC'97), Volume 2: Core and ATM Networks*, pp. 94–101.

To design low cost reliable telecommunication networks, three algorithms are proposed: an integer linear programming algorithm (branch-and-cut-and-price), a GRASP, and a zoom-in approach that combines a genetic algorithm with deterministic optimization routines. The greedy choice of the proposed GRASP is to favor paths having lowest additional cost. The local search iteratively tries to reroute some paths.

Prais, M. and C. Ribeiro (2000). Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing* **12**, 164–176.

A geostationary communication satellite has a number of spot beam antennas covering geographically distributed areas. According to the slot switching configuration on the on-board switch, the uplink traffic received at the satellite has to be immediately sent to ground areas through a set of transponders. The slot

switching configurations are determined through the solution of a time slot assignment problem, which is equivalent to the decomposition of a nonnegative traffic matrix into the sum of a family of switching mode matrices. A Reactive GRASP is proposed. See also page 358 and page 385.

Resende, L. and M. Resende (1999). A GRASP for frame relay PVC routing. In *Proc. of the Third Metaheuristics International Conference*, pp. 397–402.

The objective of the problem solved here is to minimize PVC delays while balancing trunk loads. The local search procedure reroutes each PVC, one at a time, checking each time if the new route taken together with the remaining fixed routes improves the objective function. See also page 399.

Resende, M. (1998). Computing approximate solutions of the maximum covering problem using GRASP. *Journal of Heuristics* **4**, 161–171.

A GRASP for maximum covering is proposed. Maximum covering problems arise in telecommunications network location applications. See also page 395.

Resende, M. and C. Ribeiro (2003). GRASP with path-relinking for private virtual circuit routing. *Networks* **41**, 104–114.

Variants of a GRASP with path-relinking heuristic are proposed for the offline PVC routing problem. See also page 364 and page 399.

Resende, M. and O. Ulular (1997). SMART: A tool for AT&T Worldnet access design – Location of Cascade 9000 concentrators. Technical report, AT&T Labs Research, Florham Park, NJ 07932 USA.

This report describes SMART, a software tool for finding low cost configurations of Cascade 9000 concentrators in the AT&T Worldnet backbone access network. The concentrator location problem is stated and cost model is presented for concentrator configurations. This cost model is used in a GRASP, proposed for finding approximate solutions to the concentrator location problem. The greedy choice favors the points-of-presence (POPs) with smallest incremental cost. The local search implements a simple 2-exchange.

Ribeiro, C. and I. Rosseti (2002). A parallel GRASP for the 2-path network design problem. *Lecture Notes in Computer Science* **2004**, 922–926.

A parallel GRASP with path-relinking is proposed for solving the 2-path network design problem. See also page 372 and page 378.

Rosseti, I. (2003). *Heurísticas para o problema de síntese de redes a 2-caminhos*. Ph. D. thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil.

The 2-path network design problem (2PNDP) consists in finding a minimum weighted subset of edges containing a 2-path between the endpoints of every origin-destination in D, where a 2-path between the pair $(s,t) \in D$ is a sequence of at most two edges connecting s to t. To solve 2PNDP, sequential and parallel heuristics are proposed, included variants and combinations of GRASP. In Portuguese. See also page 378.

Srinivasan, A., K. Ramakrishnan, K. Kumaram, M. Aravamudam, and S. Naqvi (2000). Optimal design of signaling networks for Internet telephony. In *IEEE INFOCOM 2000*.

An approach for efficient design of a signaling network for a network of software switches supporting Internet telephony is proposed. Optimal load balancing for given demand forecast is formulated as a quadratic assignment problem, which is solved with a GRASP.

Vieira, C. and P. Gondim (2001). Uma nova estratégia para aplicação do GRASP ao problema de alocação de canal. Technical Report 070/DE9/01, Departamento de Engenharia de Sistemas, Instituto Militar de Engenharia, Rio de Janeiro, Brazil.

A special location problem arising in telecommunications is addressed. See page 386.

Xu, J. and S. Chiu (1996). Solving a real-world field technician scheduling problem. In *Proceedings of the International Conference on Management Science and the Economic Development of China*, pp. 240–248.

The local search implements four different moves, among them the 2-exchange and a swap that exchanges an assigned job with another job unassigned under the candidate schedule. See page 405.

Xu, J. and S. Chiu (2001). Effective heuristic procedure for a field technician scheduling problem. *Journal of Heuristics* **7**, 495–509.

The objective of the field technician scheduling problem is to assign a set of jobs at different locations with time windows to technicians with different job skills. Several heuristics are designed and tested for solving the problem: a pure greedy heuristic, a GRASP, and a local search algorithm. The greedy choice of the GRASP proposed is to select jobs with the highest unit weight. The local search implements four different moves, among them the 2-exchange and a swap that exchanges an assigned job with another job unassigned under the candidate schedule. See also page 405.

13.17 ELECTRICAL POWER SYSTEMS

GRASP has been applied to problems arising in planning and operations of electrical power systems. The following papers examplify these applications.

Bahiense, L., G. Oliveira, and M. Pereira (2001). A mixed integer disjunctive model for transmission network expansion. *IEEE Transactions on Power Systems* **16**, 560–565.

Given the nonconvex nature of the transmission network expansion problem, its classical nonlinear mixed integer formulation does not guarantee an optimal solution. An alternative mixed integer linear disjunctive formulation is proposed. The mixed integer program is solved by a commercial branch and bound code, where the upper bound in the bounding phase is obtained by applying the reactive GRASP proposed in Binato et al. (2001).

Binato, S. and G. Oliveira (2002). A Reactive GRASP for transmission network expansion planning. In C. Ribeiro and P. Hansen (Eds.), *Essays and surveys in metaheuristics*, pp. 81–100. Kluwer Academic Publishers.

The GRASP previously proposed by Binato, Oliveira, and Araújo (1998) for the transmission network expansion problem is enhanced with the reactive scheme of Prais and Ribeiro (2000). A bias distribution function of Bresina (1996) is applied to bias the random greedy construction phase towards the most promising variables. See also page 357.

Binato, S., G. Oliveira, and J. Araújo (2001). A greedy randomized adaptive search procedure for transmission expansion planning. *IEEE Transactions on Power Systems* **16**, 247–253.

A GRASP for a long term transmission expansion planning problem is proposed. The greedy function minimizes the load curtailment required to eliminate all operational violations. The local search phase is based on circuit exchanges.

Faria Jr., H., S. Binato, M. Resende, and D. F. ao (2004). Power transmission network design by greedy randomized adaptive path relinking. *IEEE Transactions on Power Systems*. To appear.

A new metaheuristic based on ideas in GRASP and path relinking is applied to solve static power transmission network design problems. See also page 363.

Viana, A., J. de Sousa, and M. Matos (2003). Using GRASP to solve the unit commitment problem. *Annals of Operations Research* **120**, 117–132.

The unit commitment problem consists in deciding, over a given planning horizon, the set of electric generators to be committed and defining the production levels required for each generator so that load and spinning reserve requirements are verified at a minimum production cost. The GRASP construction phase proposed applies a greedy criterion based on fuel cost, start-up cost, and shut-down cost. The local search procedure uses a 1-flip neighborhood, where neighbors of a given solution are obtained by changing the current status of a single unit.

Vogel, I., M.-L. Flottes, and C. Landrault (2002). Initialisation des circuits séquentiels avant test intégré et scan partiel. Technical report, Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier, Montpellier, France.

The GRASP proposed by Pardalos, Qian, and Resende for the feedback set problem is applied for solving the sequential circuits initialization problem. In French.

13.18 BIOLOGY

Recent work in computational biology has applied the concepts of GRASP. The papers below illustrate this.

Bibliography

Andreatta, A. and C. Ribeiro (2002). Heuristics for the phylogeny problem. *Journal of Heuristics* **8**, 429–447.

> A phylogeny is an evolutionary tree that relates taxonomic units, based on their similarity over a set of characters. To solve the phylogeny problem is to find a phylogeny with the minimum number of evolutionary steps, i.e. applying the so-called parsimony criterion. Several heuristic approaches are studied and tested, including a GRASP. Three different neighborhood structures are investigated: nearest neighborhood interchange, the single step neighborhood, and subtree pruning and regrafting.

Andronescu, M. and B. Rastegari (2003). Motif-GRASP and Motif-ILS: Two new stochastic local search algorithms for motif finding. Technical report, Computer Science Department, University of British Columbia, Vancouver, Canada.

> A motif is a conserved pattern thought to exist in several biosequences such as DNA, RNA, and proteins. Given N biosequences S_i, $i = 1,2,\ldots,N$ with length n_i and a number L, the problem of motif finding consists in finding a sequence M_i of length L for each biosequence such that their similarity grade is maximized. A candidate solution is represented as a set $a_1, a_2, \ldots, a_N$, where $a_k \in [1, n_k - L + 1]$, for each $k \in [1, N]$. All candidate solutions correspond to all possible combinations of a_i assignment. Several greedy functions are proposed based on the a weight defined on the starting point of the motif. The neighborhood structure used in the local search procedure is a 1-exchange.

Brown, D. (2000). *Algorithmic methods in genetic mapping*. Ph. D. thesis, Cornell University, Ithaca, NY, USA.

> A survey of existing methods for genetic mapping problems is presented and several new algorithms, including a GRASP, proposed. The greedy function is defined on bin length, while the local search first removes from the sample those population members that do not affect on the objective function value.

Brown, D., T. Vision, and S. Tanksley (2000). Selecting mapping: A discrete optimization approach to select a population subset for use in a high-density genetic mapping project. *Genetics* **155**, 407–420.

A GRASP is proposed for selecting a population subset for use in a high-density genetic mapping project. At each iteration of the construction phase, one among the r unchosen population members which most improve the objective function value is added to the solution. Very small sized RCLs (i.e. $r = 3$ and $r = 5$) are used. The implemented local search removes from the current solution some members and greedily includes other members.

Fried, C., W. Hordijk, S. Prohaska, C. Stradler, and P. Stradler (2003). The footprint sorting problem. Technical report, Bioinformatics, Department of Computer Science, University of Leipzig, Germany.

Phylogenetic footprints are short pieces of no-coding DNA sequence in genes that are conserved between evolutionary distant species. It is shown that solving the footprint sorting problem requires the solution of a minimum weight vertex feedback set problem. For this the GRASP of Festa et al. (2001) is used.

Iorvik, V., E. Triantaphyllou, T. Liao, and S. Waly (1999). Predicting muscle fatigue via electromyography: A comparative study. In *Proceedings of the 25th International Conference on Computers and Industrial Engineering*, pp. 277–280.

A comparison of some state-of-the-art AI predictive and statistical techniques, including a GRASP, is presented.

Krasnogor, N., D. Pelta, W. Russo, and G. Terrazas (1998). A GRASP approach to the protein structure prediction problem. Technical report, LIFIA Lab, University of La Plata, La Plata, Argentina.

The applicability of a GRASP for solving a special protein folding problem is presented. The goal is to predict from the molecular sequence of a given protein its particular 3D structure.

Reynolds, A., J. Dicks, I. Roberts, J. Wesselink, B. de la Iglesia, V. Robert, T. Boekhout, and V. Rayward-Smith (2003). Algorithms for identification key generation and optimization with application to yeast identification. In *Applications of Evolutionary Computing*, Volume 2611 of *Lecture Notes in Computer Science*, pp. 107–118. Springer-Verlag.

For the automated creation of low cost identification keys, several algorithms are described. One of them applies the greedy randomized strategy of the GRASP framework.

Ribeiro, C. and D. Vianna (2003). A GRASP/VND heuristic for the phylogeny problem using a new neighborhood structure. Technical report, Department of Computer Science, Catholic U. of Rio de Janeiro, Rio de Janeiro, Brazil.

The phylogeny problem consists in finding a phylogeny with the minimum number of evolutionary steps, where a phylogeny is a tree that relates taxonomic units based on their similarity over a set of characters. The authors propose a hybridization of GRASP and VND. See also page 365.

13.19 VLSI DESIGN

GRASP has been used to solve circuit partitioning problems, as illustrated by the following papers.

Areibi, S. (1999). GRASP: An effective constructive technique for VLSI circuit partitioning. In *Proc. IEEE Canadian Conference on Electrical & Computer Engineering (CCECE'99)*, Volume 1, pp. 462–467.

A GRASP is proposed to obtain good initial solutions for an iterative improvement technique. See page 388 and page 393.

Areibi, S., M. Moussa, and H. Abdullah (2001). A comparison of genetic/memetic algorithms and other heuristic search techniques. In *Proceedings of IC-AI 2001*.

Several constructive procedures for circuit partitioning problems are compared, including a genetic algorithm, a memetic algorithm, and a GRASP.

Areibi, S. and A. Vannelli (1997). A GRASP clustering technique for circuit partitioning. In J. Gu and P. Pardalos (Eds.), *Satisfiability problems*, Volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pp. 711–724. American Mathematical Society.

The number of clusters is predetermined as a function of the number of partitions required. Initially, the heuristic reads the circuit description and resizes the blocks to be used by GRASP, which utilizes only the construction phase to generate the number of required clusters. The GRASP construction phase is followed by a post-processing stage, in which a simple dynamic hill climbing algorithm is used as local search to improve the initial solution generated. See also page 388.

Areibi, S. and A. Vannelli (2000). Efficient hybrid search techniques for circuit partitioning. In *IEEE 4th World Multiconference on Circuits, Systems, Communications & Computers*.

See page 362, page 388, and page 393.

13.20 AUTOMATIC DRAWING

GRASP has been used to find approximate solutions to problems related to automatic drawing. This section lists these papers.

Bibliography

Binucci, C., W. Didimo, G. Liotta, and M. Nonato (2002). Labeling heuristics for orthogonal drawings. In *Proceedings of GD'98 – Symposium on Graph Drawing*, Volume 2265 of *Lecture Notes in Computer Science*, pp. 139–153. Springer-Verlag.

Several heuristics (including a GRASP) for computing an orthogonal drawing of a graph with labels are implemented and compared.

Fernández, E. and R. Martí (1999). GRASP for seam drawing in mosaicking of aerial photographic maps. *Journal of Heuristics* **5**, 181–197.

Commercial aerial photographic maps are often so large that it is necessary to produce one map from two or even more photographs. These are combined, two at a time, in a process called *mosaicking*. The most difficult step in the mosaicking process is *seam-drawing*. A GRASP is proposed for solving the seam-drawing process.

Laguna, M. and R. Martí (1999). GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing* **11**, 44–52.

A GRASP with path-relinking is developed for the problem of minimizing straight line crossings in a 2-layer graph. The greedy criterion of the construction phase is based on the degree of the vertices and a value based restricted candidate list is used. Each step of the improvement phase consists in selecting each vertex to be considered for a move. A probabilistic selection rule is used such that vertices with high degree are more likely to be selected first at each step of this process. See also page 357 and page 363.

Martí, R. and V. Estruch (2001). Incremental bipartite drawing problem. *Computers and Operations Research* **28**, 1287–1298.

A GRASP is proposed for the incremental arc crossing minimization problem for bipartite graphs. Computational experiments are done on 450 instances and results are compared with a branch and bound algorithm. See also page 376.

Martí, R. and M. Laguna (2003). Heuristics and meta-heuristics for 2-layer straight line crossing minimization. *Discrete Applied Mathematics* **127**, 665–678.

Extensive computational results are presented using 12 heuristics and two meta-heuristics for the 2-layer straight line crossing minimization problem. On dense

graphs, a tabu search meta-heuristic does best with GRASP a close second. On low-density graphs, GRASP outperforms all other approaches. See also page 376.

Ng, K. and B. Trifonov (2003). Automatic bounding volume hierarchy generation using stochastic search methods. In *CPSC532D Mini-Workshop "Stochastic Search Algorithms"*.

A bounding volume hierarchy is used for improving the efficiency of ray tracing based rendering. Finding good hierarchies is difficult, since the number of hierarchies grows exponentially with the number of scene objects. A GRASP is designed for improving previously proposed heuristics. The greedy function used is based on subdivision points, while local search is basically a perturbation procedure.

Osman, I., B. Al-Ayoubi, and M. Barake (2003). A greedy random adaptive search procedure for the maximal planar graph problem. *Computers and Industrial Engineering* **45**, 635–651.

A GRASP is proposed and tested for the weighted maximal planar graph problem. The construction is a randomized version of the Green and Al-Hakim algorithm (1985). A new data structure is introduced, reducing the complexity of the construction from $O(n^3)$ to $O(n^2)$. Local search uses four types of moves proposed by Pesch, Glover, Bartsch, Salewski, and Osman (1995). See also page 377.

Osman, I., B. Al-Ayoubi, M. Barake, and M. Hasan (2000). A greedy random adaptive search procedure for the weighted maximal planar graph problem. Technical report, School of Business and Center for Advanced Mathematical Sciences, American University of Beirut, Beirut, Lebanon.

A GRASP is proposed and tested for the weighted maximal planar graph problem. See also page 377.

Osman, I., M. Hasan, and A. Abdullah (2002). Linear programming based meta-heuristics for the weighted maximal planar graph. *Journal of the Operational Research Society* **53**, 1142–1149.

Two meta-heuristics are described, both derived from an ILP relaxation. The first one takes into account only variables with fractional value greater than half in the ILP relaxation to build an initial subgraph from which a planar subgraph is extracted with the help of a GRASP and triangulation of faces. The second approach considers only edges having integer value in the ILP relaxation, while the remaining edges are sorted in descending order of their weights. Those edges that do not violate a planarity test are thus candidate for insertion to obtain a feasible solution using GRASP. See also page 377.

Resende, M. and C. Ribeiro (1997). A GRASP for graph planarization. *Networks* **29**, 173–189.

A GRASP is described that extends the two-phase heuristic of Goldschmidt and Takvorian (*Networks*, v. 24, pp. 69–73, 1994). Computational experience on a large set of standard test problems is presented. On almost all test problems considered, the heuristic either matches or finds a better solution than previously

described graph planarization heuristics. In several cases, previously unknown optimal solutions are found. See also page 378.

Ribeiro, C. and M. Resende (1999). Algorithm 797: Fortran subroutines for approximate solution of graph planarization problems using GRASP. *ACM Transactions on Mathematical Software* **25**, 341–352.

A set of Fortran subroutines that implements the GRASP for graph planarization of Resende and Ribeiro (1997) is presented. See also page 368 and page 378.

13.21 MISCELLANEOUS

The papers in this section could not be categoried into any of the previous section in this paper.

Boeres, M., C. Ribeiro, and I. Bloch (2003). Randomized algorithms for scene recognition by inexact graph matching. Technical report, Computer Science Department, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil.

The algorithm proposed to solve a scene recognition problem consists of a randomized construction procedure and a local search procedure. In the construction procedure, the greedy function has two terms representing, respectively, the node and edge contributions to the measure of the solution quality associated with the correspondence. Given a feasible solution x, the neighborhood structure used during local search considers as neighbors of x all feasible solutions that can be obtained by changing some association. See page 374.

cois, H. F. and O. Boëffard (2002). The greedy algorithm and its application to the construction of a continuous speech database. In *Proceedings of LREC-2002*, Volume 5, pp. 1420–1426.

A general framework of the construction of databases characterized by different linguistic features is addressed. A small sized continuous speech database is needed, at the same time based on a maximum number of phonetic units. After recorded, the set of sentences constitutes the source from which the text-to-speech synthesizer draws the needed acoustic units. The problem is to find the smallest subset of sentences that covers all needed units. A greedy algorithm is described to solve the problem and the development of a GRASP is proposed as future work.

de Noronha, T. F. (2001). Algoritmos e estratégias de solução para o problema do gerenciamento de sondas de produção terrestre na bacia petrolífera potiguar. *Revista Electrônica de Iniciação Cientifica* **1**.

Several different heuristics are proposed, including a GRASP, for a petroleum production planning problem. The greedy function is a simple cost function related to the production, while the local search phase looks for improving solutions by swapping paths. In Portuguese.

de Souza, C. C., C. B. Medeiros, and R. S. Pereira (1996). Integrating heuristics and spatial databases: A case study. Technical Report IC-96-18, Institute of Computing, Universidade Estadual de Campinas, Campinas, Brazil.

Part of the ongoing efforts at IC-UNICAMP to apply heuristic algorithms to vectorial georeferenced data to help decision support in urban planning is described. A first prototype, implemented in C++, and tested on support planning activities for the São Paulo State Post Office System in Brazil is presented. The problem is a special partition problem, where the number of clusters in the partition (number of districts in the distribution zone) must be minimized. The problem is represented by building a special undirected graph that has two main characteristics: connectivity and information about the mailman daily loads. To solve the problem, a set of randomized heuristics, including a GRASP, are proposed.

Demirer, R. and B. Eksioglu (1998). Subset selection in multiple linear regression: A new mathematical programming approach. Technical Report School of business working paper no. 284, School of business, University of Kansas, Lawrence, Kansas, USA.

A new mathematical programming model is proposed. It is parametrically solved to obtain a collection of efficient subsets. The parametric solution requires repeatedly solving a mathematical program which is done with either a Lagrangian relaxation based heuristic or a GRASP.

Festa, P. and G. Raiconi (2001). GRASP in switching input optimal control synthesis. In *Proceedings of MIC'2001*, pp. 381–385.

Several optimal control problems are introduced. A GRASP is designed. In the construction phase, the greedy criterion minimizes the quadratic costs in the Riccati equation. The neighborhood structure used in the local search phase is defined on the Hamming distance.

Gandibleux, X., D. Vancoppenolle, and D. Tuyttens (1998). A first making use of GRASP for solving MOCO problems. Technical report, University of Valenciennes, France.

An extension of GRASP, to solve multi-objective combinatorial optimization (MOCO) problems, is considered. In particular, classical covering, assignment, knapsack, and scheduling problems with multiple objectives are used as benchmarks. Computational results compare GRASP solutions for a benchmark set of test problems and results are discussed in comparison with an exact method, when available. In French.

Ghosh, J. (1996). Computational aspects of the maximum diversity problem. *Operations Research Letters* **19**, 175–181.

Two variations of the maximum diversity problem are addressed. This problem arises when m elements are to be selected from an n-element population based on inter-element distances. Using a reduction from the vertex cover problem, a GRASP is proposed.

Gomes, A. and J. Oliveira (2001). A GRASP approach to the nesting problem. In *Proceedings of MIC'2001*, pp. 47–52.

A GRASP is proposed to solve a variant of the classical nesting problem, whose objective is to minimize the length of a single plate used to produce a given set of smaller pieces. Two greedy criteria are tested. The first one is the layout length, measured as the maximum coordinate in the current layout, while the second one

is the added internal waste, measured as the potential area lost when placing one piece. The local search phase uses neighbors obtained by exchanging pairs of pieces in the sequence output of the construction phase.

Juillé, H. and J. Pollack (1998). A sampling-based heuristic for tree search applied to grammar induction. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*.

SAGE, a new search algorithm that incorporates the same fundamental mechanisms as the most popular metaheuristics, is presented. It is an iterative search procedure that at each iteration performs a construction phase and a competition phase. In the construction phase, SAGE implements a set of elementary randomized searches and a meta-level heuristic that controls the search procedure by distributing the alternatives among the searches. Scope of the competition phase is to favor the most promising search alternatives.

Locatelli, M. (1998). A class of heuristic methods for the maximization of the l_1-norm over parallelotopes. Technical report, Dipartimento di Sistemi ed Informatica, Universitá di Firenze, Firenze, Italy.

For the maximization of the l_1-norm over parallelotopes, a class of heuristics is proposed that includes a slightly modified GRASP, in which between a greedy construction and local search phases a filter phase is inserted to avoid performing local search from bad starting solutions. The local search uses is a variant of a 1-flip neighborhood.

Medeiros, M., M. Resende, and A. Veiga (2001). Piecewise linear time series estimation with GRASP. *Computational Optimization and Applications* **19**, 127–144.

A GRASP is proposed to build piecewise linear statistical models with multivariate thresholds. The construction phase consists of sequentially choosing hyperplanes until the maximum number of hyperplanes is reached. The greedy function orders the possible hyperplanes with respect to the sum of squared errors of the fitted data. The local search is a 2-exchange heuristic.

Medeiros, M., A. Veiga, and M. Resende (2002). A combinatorial approach to piecewise linear time analysis. *Journal of Computational and Graphical Statistics* **11**, 236–258.

A new approach to modeling threshold processes is proposed. It is based on a linear model with time-varying parameters. This formulation is shown to be closely related to the self-exciting threshold autoregressive models (SETAR) with the advantage that it incorporates linear multivariate thresholds. A GRASP is proposed to estimate the parameters of the model. The greedy choice takes into account the sum of squared errors of the fitted data. The local search is a 2-exchange heuristic.

Mockus, J., E. Eddy, A. Mockus, L. Mockus, and G. Reklaitis (1997). *Bayesian discrete and global optimization*. Kluwer Academic Publishers.

This book describes the Bayesian approach to discrete optimization. A Bayesian heuristic algorithm version of GRASP is described.

Neto, T. and J. Pedroso (2001). GRASP for linear integer programming. In *Proceedings of MIC'2001*, pp. 377–380.

The GRASP framework is extended for solving general linear integer problems. The key is to split the variables into a set of integer and a set of linear variables. Then, GRASP finds values of the integer variables that are replaced in the original problem, which becomes a pure continuous problem solvable by any linear programming solver.

Neto, T. and J. Pedroso (2003). GRASP for linear integer programming. In M. Resende and J. de Sousa (Eds.), *Metaheuristics: Computer decision-making*, pp. 545–574. Kluwer Academic Publishers.

A preliminary version in this paper appeared in Neto and Pedroso (2001). Here, the GRASP framework is extended for solving general linear integer problems. The key is to split the variables into a set of integer and a set of linear variables. Then, GRASP finds values of the integer variables that are replaced in the original problem, which becomes a pure continuous problem, solvable by any linear programming solver.

Palubeckis, G. and A. Tomkevicius (2002). GRASP implementations for the unconstrained binary quadratic optimization problem. *Information Technology and Control* **24**, 14–20.

A classical GRASP framework and an enhanced GRASP that uses a simple tabu search as local search are proposed. Numerical results show that the enhancement introduced in the classical GRASP implementation produces higher quality solutions. See also page 364.

Roli, A. and M. Milano (2002). MAGMA: A multiagent architecture for metaheuristics. Technical Report DEIS-LIA-02-007, DEIS, Università degli Studi di Bologna, Bologna, Italy.

The main metaheuristic schemes, including GRASP, are revisited in a multiagent perspective and a uniform framework called MAGMA is provided.

Sellmann, M. and W. Harvey (2002). Heuristic constraint propagation using local search for incomplete pruning and domain filtering of redundant constraints for the social golfer problem. In *CPAIOR 2002)*.

It is hard to efficiently establish the existence of a feasible solution in a constraint satisfaction problem. The most common approach is to implicitly explore the feasible region. This paper proposes to add tight redundant constraints, possibly hard to be verified exactly, but that can be checked by applying heuristics. One heuristic strategy considered follows a GRASP strategy.

Silva, G., L. Ochi, and S. Martins (2003). Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem. Technical report, Department of Computer Science, Federal Fluminense University, Niterói, Brazil.

Given a large collection C of elements, the maximum diversity problem consists in finding optimally diverse subsets of C. Ghosh (1996) proposed a GRASP for approaching this problem. Here, a new GRASP is depicted, whose construction phase implements three different strategies based on distance greedy function. The local search phase uses two neighborhood structures. One is the structure defined by Ghosh, while the second one is a 2-exchange neighborhood.

13.22 RECENT PUBLICATIONS

In this section, we list some recent publications. These will later be reclassified into the previous sections.

Bibliography

Abello, J., M. Resende, and S. Sudarsky (2002). Massive quasi-clique detection. In S. Rajsbaum (Ed.), *LATIN 2002: Theoretical Informatics*, Volume 2286 of *Lecture Notes in Computer Science*, pp. 598–612. Springer-Verlag.

> The authors propose techniques that are useful for the detection of dense subgraphs (quasi-cliques) in massive sparse graphs whose vertex set, but not the edge set, fits in RAM. The algorithms rely also on greedy randomized adaptive search procedures (GRASP) to extract the dense subgraphs.

.

Aiex, R., S. Binato, and M. Resende (2003). Parallel grasp with path-relinking for job shop scheduling. *Parallel Computing* **29**, 393–430.

> This paper describes a parallel GRASP with path relinking. Independent and cooperative parallelization strategies are described and implemented. Two greedy functions are defined: the first on is the makespan resulting from the inclusion of a candidate operation to the already-scheduled operations; while the second one (used in conjunction with the makespan) favors operations from jobs having long remaining processing times. RCL is built by applying a min-max α-percentage rule. The authors employ the two exchange local search.

.

Aiex, R., P. Pardalos, L. Pitsoulis, and M. Resende (2000). A GRASP for computing approximate solutions for the three-index assignment problem. In *Proceedings of Parallel and Distributed Processing*, Volume 1080 of *Lecture Notes in Computer Science*, pp. 504.

> The GRASP construction phase builds a feasible solution *S* by selecting *n* triplets one at a time and the greedy choice reflects the weight associated with each ordered triplet. In the local search a 2-exchange neighborhood is adopted.

.

Aiex, R. and M. Resende (2005). Parallel strategies for grasp with path-relinking. In T. Ibaraki, K. Nonobe, and M. Yagiura (Eds.), *Metaheuristics: Progress as Real Problem Solvers*. Springer.

The authors analyze two parallel strategies for GRASP with path-relinking and propose a criterion to predict parallel speedup based on experiments with a sequential implementation of the algorithm. Independent and cooperative parallel strategies are described and implemented for the 3-index assignment problem and the job-shop scheduling problem.

.

Aiex, R., M. Resende, and C. Ribeiro (2006). Tttplots: A perl program to create time-to-target plots. *Optimization Letters, published online*.

This papers describes a Perl language program to create time-to-target solution value plots for measured CPU times that are assumed to fit a shifted exponential distribution, as in the case of randomized local search based heuristics for combinatorial optimization. The authors show how to use such plots in the comparison of different algorithms or strategies for solving a given problem. A detailed description of the Perl program `tttplots.pl` is also provided.

.

Alvarez-Valdes, R., F. Parreo, and J. Tamarit (2005). A grasp algorithm for constrained two-dimensional non-guillotine cutting problems. *Journal of the Operational Research Society* **56**(4), 414–425.

A GRASP is designed for the constrained two-dimensional non-guillotine cutting problem, which consists in cutting the rectangular pieces from a large rectangle so as to maximize the value of the pieces cut. The authors investigate several strategies for the construction and improvement phases. In the construction phase, the authors take the smallest rectangle breaking the ties by the nearest distance to a corner of the stock rectangle. Then, two criteria have been considered to select the piece: 1) The first piece in a list ordered, giving priority to pieces which must be cut; 2) The piece producing the largest increase in the objective function. Three alternative local search procedures are proposed: 1) A block adjacent to an empty rectangle is selected and a tentative of completely eliminating it is performed. The remaining pieces are then moved to the corners, the empty rectangles are merged and the resulting list of empty rectangles is filled by applying the constructive algorithm. 2) The second procedure is a simplification of method 1) in which pieces are not moved to the corners and the new empty rectangles are only merged with existing adjacent empty rectangles. 3) The third procedure consists of eliminating the final $k\%$ blocks of the solution and filling the empty space with the deterministic constructive algorithm. Once the final pieces have been removed from the solution, the remaining pieces are moved to the corners, the empty rectangles are merged and the constructive algorithm is then applied.

.

Alvarez-Valdes, R., F. Parreo, and J. Tamarit (2007). Reactive grasp for the strip-packing problem. *Computers & Operations Research, to appear*, –.

This paper proposes a GRASP for the strip packing, where the problem is to place a set of rectangular pieces into a strip of a given width and infinite length so as to minimize the required length. The n pieces to pack are grouped into m types of pieces of dimensions (h_i, w_i), $i = 1, \ldots, m$, and it is needed to pack Q_i

copies of each type i, with $\sum_{i=1}^{m} Q_i = n$. The greedy choice is to sort the set of piece types still to be packed in non-increasing order respect to w_i and ties are broken by non-increasing h_i. The authors propose several different construction procedures. The first three criteria are based on the width, trying to fill the bottom of rectangle as much as possible. Each one of them gives a different importance to the height of the pieces. The fourth criterion tries to maintain a profile of the current solution which is as smooth as possible, avoiding peaks and troughs. Several local searches are described. In the first one, from the initial solution of height H, the authors define a closed stock sheet of width W and height $H - 1$ and remove the last $k\%$ pieces from the solution. In the second one, the pieces defining the maximum height H are removed from the solution and placed on some of the waste rectangles at lower levels of the strip. The third method consists of eliminating the last $k\%$ pieces of the solution and filling the empty space with the deterministic constructive algorithm. The fourth method is similar to the third one, but in this case all pieces with their upper side exceeding a height λH are removed, with $0 < \lambda < 1$.

.

Amaldi, E., A. Capone, and F. Malucelli (2003). Planning umts base station location: Optimization models with power control and algorithms. *IEEE Transactions on Wireless Communications* **2**(5), 939–952.

In this paper, the authors propose discrete optimization models and algorithms aimed at supporting the decisions in the process of planning where to locate new BSs. As the authors underline, these models consider the signal-to-interference ratio as quality measure and capture at different levels of detail the signal quality requirements and the specific PC mechanism of the wideband CDMA air interface. Two randomized greedy procedures and a TS algorithm for the uplink (mobile to BS) direction are described. In particular, the authors devise a greedy randomized procedure and a reverse greedy randomized procedure that construct a solution, i.e. a subset of candidate sites where to activate BSs by iteratively adding and removing BSs from the current solution, respectively. The greedy criterion takes into account the number of connections they could service. The authors define swap moves that amount to installing a new BS in one of the empty sites while deleting one of the active BSs by only considering swaps between candidate sites that are relatively close to each other.

.

Amaldi, E., A. Capone, F. Malucelli, and F. Signori (2003). Optimization models and algorithms for downlink umts radio planning. In *Proceedings of Wireless Communications and Networking, (WCNC 2003)*, Volume 2, pp. 827–831.

The authors describe two mathematical programming models for locating directive base stations considering downlink (base station to mobile) direction and assuming a power-based as well or a SIR-based power control mechanism. To efficintly solve the NP-hard downlink BS locationthe problem, a GRASP and a TS are proposed. Each GRASP construction phase starts from an empty set of active BSs and at each iteration randomly selects an available CS (in which to install an additional BS) from a set of available CSs which yield the best improvements in the objective function. In the local search the following moves are considered to explore the solution space: removing a BS, installing a new BS,

removing an existing BS and installing a new one (swap). The output GRASP solution is used ss initial solution for a TS algorithm.

.

Andrade, D. and M. Resende (2006). A GRASP for PBX telephone migration scheduling. In *Proceedings of the Eighth INFORMS Telecommunications Conference*, pp. –.

A PBX, or private branch exchange, is a private telephone network. The PBX telephone migration problem arises when an enterprise acquires a new PBX to replace an existing one. Phone numbers need to migrate from the old system to the new system over a time horizon. A penalty, assigned to the each pair of phones, is incurred if the pair is migrated in different time periods. The objective is to assign phones to time periods such that no more than a given number of phones is assigned to any period and the total penalty is minimized. The authors propose a GRASP for approximatively solve this problem, where a solution is an assignment of phone numbers to time periods such that each time period has no more than a fixed telephone numbers assigned to it. The construction procedure sequences the phone numbers and assigns them evenly to each time period. The greedy criterion is to minimize a penalty function associated with migrating phone numbers in different time periods. Once a feasible solution is constructed, local search is applied using three neighborhoods: swap phones, move phone, and swap periods.

.

Andrade, D. and M. Resende (2007). Grasp with evolutionary path-relinking. Technical report, AT&T Labs Research Technical Report TD-6XPTS7, Florham Park, NJ, USA.

The authors propose GRASP with evolutionary path-relinking (EvPR), a metaheuristic resulting from the hybridization of GRASP, path-relinking, and evolutionary path-relinking. The new proposed hybrid framework is applied to a network migration problem and experiments show that it is able to find good approximate solutions faster than a heuristic based on GRASP with path-relinking as well as one based on pure GRASP. In EvPR, the solutions in the pool are evolved as a series of populations P_1, P_2, ... of equal size. The initial population is the pool of elite solutions produced by GRASP with PR. In iteration k of EvPR, path-relinking is applied between a set of pairs of solutions in population P_k and, with the same rules used to test for membership in the pool of elite solutions, each resulting solution is tested for membership in population P_{k+1} . This evolutionary process is repeated until no improvement is seen from one population to the next.

.

Andreatta, A. and C. Ribeiro (2002). Heuristics for the phylogeny problem. *Journal of Heuristics* **8**, 429–447.

A phylogeny is a tree that relates taxonomic units, based on their similarity over a set of characters. The phylogeny problem consists in finding a phylogeny with the minimum number of evolutionary steps (the so-called parsimony criterion). The authors propose different heuristic approaches to the phylogeny problem

under the parsimony criterion, including a GRASP and a VNS. The greedy randomized construction randomly selects a pair taxon-branch from among all those leading to the most parsimonius increment value. The local search phase is implemented with the selection of the first improving move. Two neighborhood strategies are used to devise two alternative GRASP algorithms. The first one applies subtree pruning and regrafting (SPR): solutions are obtained by eliminating one internal node and its three adjacent branches. Next, two pending nodes are joined by a new branch. The still pending subtree is reconnected by its pending node to a branch of the other subtree. The second one defines three different neighborhoods explored within a VND procedure: SPR, a nearest neighborhood interchanges (NNI), and the single step neighborhood (STEP), where a neighbor is obtained by taking out a taxon (i.e., a leave) from the current solution and putting it back into another branch of the tree.

.

Boudia, M., M. Louly, and C. Prins (2007). A reactive grasp and path relinking for a combined production-distribution problem. *Computers and Operations Research* **34**, 3402–3419.

In this article, an NP-hard production-distribution problem for one product over a multi-period horizon is studied. The aim is to minimize total cost taking production setups, inventory levels, and distribution. A GRASP and two improved versions using either a reactive mechanism or a path relinking are proposed. The greedy criterion takes into account the best insertion position and the minimum associated cost. The GRASP construction phase performs two steps. At the first step, it computes day by day a preliminary production plan and associates trips without storage at the plan. Then, during the second step, it tries to shift some production day to achieve the best compromise between setup and storage costs at the plan. Local search considers different types of moves that for each costumer in a day change the quantity delivered, the day of production, the day of delivery, the delivery trip, and the position in this trip.

.

Canuto, S., M. Resende, and C. Ribeiro (2001). Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks* **38**, 50–58.

In this paper, the authors describe a multi-start local search algorithm for the prize-collecting Steiner tree problem, based on the generation of initial solutions by a primal-dual algorithm using perturbed node prizes. Path relinking is used to improve the solutions found by local search and variable neighborhood search is used as a post-optimization procedure. The neighborhood of a solution is formed by all minimum spanning trees whose sets of nodes differ by exactly one node. The proposed perturbation algorithm is similar to a GRASP procedure, in which the greedy randomized construction is replaced by the construction of initial solutions using perturbed cost functions.

.

Commander, C., S. Butenko, P. Pardalos, and C. Oliveira (2004). Reactive GRASP with path relinking for the broadcast scheduling problem. In *Proceedings of the 40th Annual International Telemetry Conference*, pp. 792–800.

The Broadcast Scheduling Problem (BSP) is an NP-complete problem that arises in the study of wireless networks. A finite set of stations are to be scheduled in a time division multiple access (TDMA) frame with the objective of finding a collision free transmission schedule with the minimum number of TDMA slots and maximal slot utilization. Such a schedule must also minimize the total system delay. The authors propose variations of a GRASP with path relinking as post-optimization strategy. They also describe a reactivity method to balance GRASP parameters. The greedy choice consists in soritng the stations in descending order of the number of one-hop and two-hop neighbors. Starting from the schedule output of the construction phase, the local search procedure sorts the slots in descending order of the number of bursts. The two slots with the fewest transmissions are combined. A colliding station from the combined slot is then randomly selected and every attempt is made to swap this station with another from the remaining slots.

.

Commander, C., P. Festa, C. Oliveira, P. Pardalos, M. Resende, and M. Tsitselis (2006a). A greedy randomized algorithm for the cooperative communication problem on ad hoc networks. In *Proceedings of the Eighth INFORMS Telecommunications Conference*, pp. –.

Ad hoc networks are composed of a set of wireless units that can communicate without the use of a pre-established server infrastructure. Each client has the capacity of accessing network nodes that are within its reach. The problem consists of maximizing the amount of connectivity among a set of users, subject to constraints on the maximum distance traveled, as well as restrictions on what types of movement can be performed. The greedy function value of each candidate element is a measure of additional connections created by its insertion in the partial solution under construction. The local search procedure is based on a perturbation function consisting of selecting a wireless agent and rerouting.

.

Commander, C., P. Festa, C. Oliveira, P. Pardalos, M. Resende, and M. Tsitselis (2006b). Grasp with path-relinking for the cooperative communication problem on ad hoc networks. Technical report, AT&T Labs Research Technical Report TD-6X3U73, Florham Park, NJ, USA.

Ad hoc networks are composed of a set of wireless units that can communicate without the use of a pre-established server infrastructure. Each client has the capacity of accessing network nodes that are within its reach. The problem consists of maximizing the amount of connectivity among a set of users, subject to constraints on the maximum distance traveled, as well as restrictions on what types of movement can be performed. The greedy function value of each candidate element is a measure of additional connections created by its insertion in the partial solution under construction. The local search procedure is based on a perturbation function consisting of selecting a wireless agent and rerouting.

.

Commander, C., C. Oliveira, P. Pardalos, and M. Resende (2005). A GRASP heuristic for the cooperative communication problem in ad hoc networks. In *Proceedings of the VI Metaheuristics International Conference (MIC2005)*, pp. 225–330.

This paper is a preliminary version of 13.22.

.

Cotta, C. and A. Fernndez (2004). A hybrid grasp evolutionary algorithm approach to golomb ruler search. In *Parallel Problem Solving from Nature - PPSN VIII*, Volume 3242 of *Lecture Notes in Computer Science*, pp. 481–490. Springer-Verlag.

The point-feature cartographic label placement problem (PFCLP) is an NP-hard problem whose main application involves maps production. The labels must be placed in predefined places avoiding overlaps and considering cartographic preferences. Due to its high complexity, the scientific community has proposed and implemented several heuristics searching for approximated solutions. This paper proposes a GRASP that takes as input the conflict graph associated with the problem. Considering the PFCLP represented by a conflict graph, the constructive phase greedy criterion is based in the vertex degrees, since the degree of a vertex is a measure of labels in conflict. To build the RCL a fixed-size criterion is adopted. Starting from the built solution, the local search procedure tests for each point another valid candidate position and performs the best change. Computational results show that GRASP generates better solutions than all those reported in the literature in reasonable computational times.

.

Delorme, X., X. Gandibleux, and J. Rodriguez (2003). Grasp for set packing problems. *European Journal of Operational Research* **153**(3), 564–580.

A GRASP is described for the set packing problem. The proposed framework uses a self-tuning procedure (reactive GRASP), an intensification procedure (using path relinking), and a procedure involving the diversification of the selection (using a learning process). A set of test problem instances includes real-world railway planning instances, never solved before by means of metaheuristics.

.

Duarte, A. and R. Martí (2007). Tabu search and grasp for the maximum diversity problem. *European Journal of Operational Research* **178**(1), 71–84.

In this paper, the authors propose a new heuristic based on the tabu search methodology and incorporating memory structures for both construction and improvement. They compare their tabu search construction with a memory-less design and with previous algorithms recently developed for this problem. The constructive method can be coupled with a local search procedure or a short-term tabu search for improved outcomes.

.

e D.S. Vianna, C. R. (2005). A grasp/vnd heuristic for the phylogeny problem using a new neighborhood structure. *International Transactions in Operational Research* **12**, 325–338.

A phylogeny is a tree that relates taxonomic units, based on their similarity over a set of characters. The phylogeny problem consists in finding a phylogeny with the minimum number of evolutionary steps. The authors propose a GRASP that uses variable neighborhood descent for local search. At a generic construction phase iteration, a pair taxon-branch is randomly selected from among all those

with cost 10% higher than the most parsimonious increment value. The neighborhood defined is the subtree pruning and regrafting (SPR or 1-SPR): a subtree of the current tree is disconnected and reconnected in a different position.

.

e S. Urrutia, C. R. (2007). Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research* **179**, 775–787.

The Traveling Tournament Problem models some sport timetabling issues, where the objective is to minimize the total distance traveled by the teams. In this work, the authors study the mirrored version of the problem. They propose a fast constructive algorithm and a new heuristic based on the combination of GRASP and Iterated Local Search metaheuristics. A strong neighborhood based on ejection chains is also proposed.

.

Ekisoglu, S. D., P. Pardalos, and M. Resende (2002). Parallel metaheuristics for combinatorial optimization. In R. C. et al. (Ed.), *Models for Parallel and Distributed Computation - Theory, Algorithmic Techniques and Applications*, pp. 179–206. Kluwer Academic Publishers.

Parallel metaheuristics for approximating hard combinatorial optimization problems are reviewed. Recent developments on parallel implementation of genetic algorithms, simulated annealing, tabu search, variable neighborhood search, and greedy randomized adaptive search procedures (GRASP) are discussed.

.

Festa, P. (2007). On some optimization problems in molecular biology. *Mathematical Bioscience* **207**(2), 219–234.

This paper provides a detailed description of some among the most interesting molecular biology problems that can be formulated as combinatorial optimization problems and proposes a GRASP to find improved solutions for a particular class of them, known as the far from most string problem. The greedy function takes into account the contribution to the objective function achieved by selecting a particular element. In the case of the far from most string problem, it is intuitive to relate the greedy function to the occurrence of each character in a given position. To realize the local search phase the 2-exchange algorithm is used.

.

Festa, P., P. Pardalos, L. Pitsoulis, and M. Resende (2005). GRASP with path-relinking for the weighted maximum satisfiability problem. In *Proceedings of the IV Workshop on Efficient and Experimental Algorithms (WEA2005)*, Volume 3503 of *Lecture Notes in Computer Science*, pp. 367–379.

A GRASP with path relinking for finding good quality solutions of the weighted maximum satisfability problem (MAX-SAT) is described in this paper. Here, path relinking is used to intensify the search around good quality isolated solutions that have been produced by the GRASP heuristic. Experimental comparison of the pure GRASP (without path relinking) and the GRASP with path relinking illustrates the effectiveness of path relinking in decreasing the average time needed to find a good quality solution. In the construction phase, given any

partial solution, corresponding to a set of satisfied clauses, the next candidate element to be added to the solution to maximize the total weight of the unsatisfied clauses that become satisfied after the assignment. Once obtained a completed a truth assignment, local search is applied using the simple 1-flip neighborhood.

.

Festa, P., P. Pardalos, L. Pitsoulis, and M. Resende (2006). Grasp with path-relinking for the weighted maxsat problem. *ACM J. of Experimental Algorithmics* **11**, 1–16.

A GRASP with path relinking for finding good quality solutions of the weighted maximum satisfability problem (MAX-SAT) is described in this paper. Construction and local search phase details are as in 13.22. In this paper, the authors designed accurate experiments designed to determine the effect of path relinking on the convergence of the GRASP for MAX-SAT. Since GRASP and GRASP with PR are both stochastic local search algorithms, the authors compare their performance by examining the distributions of their running times. The results show that: 1) given a fixed amount of computing time, GRASP with PR has a higher probability than GRASP of finding a target solution; 2) given a fixed probability of finding a target solution, the expected time taken by GRASP to find a solution with that probability is greater than the time taken by GRASP with PR.

.

Festa, P., P. Pardalos, M. Resende, and Ribeiro (2001). GRASP and VNS for Max-Cut. In J. Sousa (Ed.), *Proceedings of the IV Metaheuristics International Conference (MIC2001)*, pp. 371–376.

In this paper, a GRASP and a variable neighborhood search (VNS) for MAX-CUT are proposed and tested. In the case of the MAX-CUT problem, it is intuitive to relate the greedy function to the sum of the weights of the edges in each cut. Local search is based in Boolean flip operations.

.

Festa, P., P. Pardalos, M. Resende, and C. Ribeiro (2002). Randomized heuristics for the max-cut problem. *Optimization Methods and Software* **7**, 1033–1058.

In this paper, a GRASP, a variable neighborhood search (VNS), and a path relinking as intensification heuristic for MAX-CUT are proposed and tested. New hybrid heuristics that combine GRASP, VNS as GRASP local search, and path relinking are also proposed and tested. In the case of the MAX-CUT problem, it is intuitive to relate the greedy function to the sum of the weights of the edges in each cut. Local search is based in Boolean flip operations. On a set of standard test problems, new best known solutions were produced for many of the instances.

.

Festa, P. and M. Resende (2002). GRASP: An annotated bibliography. In C. Ribeiro and P. Hansen (Eds.), *Essays and Surveys on Metaheuristics*, pp. 325–367. Kluwer Academic Publishers.

Since 1989, numerous papers on the basic aspects of GRASP, as well as enhancements to the basic metaheuristic have appeared in the literature. GRASP

has been applied to a wide range of combinatorial optimization problems, ranging from scheduling and routing to drawing and turbine balancing. This paper is an annotated bibliography of the GRASP literature from 1989 to 2001.

.

Han, B. and V. Raja (2003). A grasp heuristic for solving an extended capacitated concentrator location problem. *International Journal of Information Technology and Decision Making* **2**(4), 597–617.

In the GRASP construction phase, the greedy criterion looks for the minimum number of concentrators required to obtain a feasible solution. Two local search procedures are proposed and tested. The first one tries to eliminate a concentrator with low capacity consumption by trying to reassign its end-user nodes to other concentrators. The second one tries for each concentrator to replace it by a cheaper one while not violating capacity constraints.

.

Hirsch, M., C. Meneses, P. Pardalos, M. Ragle, and M. Resende (2006). A continuous grasp to determine the relationship between drugs and adverse reactions. Technical report, AT&T Labs Research Technical Report TD-6UPR92, Florham Park, NJ, USA.

Adverse drug reactions (ADRs) are estimated to be one of the leading causes of death. The authors formulate the drug-reaction relationship problem as a continuous optimization problem and utilize continuos GRASP (C-GRASP), a new continuous global optimization heuristic, to approximately determine the relationship between drugs and adverse reactions. C-GRASP works by discretizing the domain into a uniform grid. Both the construction and local improvement phases move along points on the grid. As the algorithm progresses, the grid adaptively becomes more dense. RCL is formed containing unfixed coordinates whose objective function value is better or equal to a threshold value computed by applying the min-max α-percentage rule. The local improvement phase (with pseudo-code can be seen as approximating the role of the gradient of the objective function.

.

Hirsch, M., C. Meneses, P. Pardalos, and M. Resende (2007). Global optimization by continuous grasp. *Optimization Letters* **1**(2), 201–212.

In this paper a new global optimization method, called Continuous GRASP (C-GRASP), is proposed. It extends Feo and Resende's GRASP from the domain of discrete optimization to that of continuous global optimization and is well-suited approach for solving global optimization problems, since it does not make use of derivative information. C-GRASP resembles GRASP in that it is a multistart stochastic search metaheuristic that uses a randomized greedy procedure to generate starting solutions for a local improvement algorithm. The main difference is that an iteration of C-GRASP does not consist of a single greedy randomized construction followed by local improvement, but rather a series of construction-local improvement cycles with the output of construction serving as the input of the local improvement, as in GRASP, and the output of the local improvement serving as the input of the construction procedure, unlike GRASP. C-GRASP

works by discretizing the domain into a uniform grid. Both the construction and local improvement phases move along points on the grid. As the algorithm progresses, the grid adaptively becomes more dense.

.

Hirsch, M., P. Pardalos, and M. Resende (2006a). Recognition of projected 3d points and lines using a continuous grasp. Technical report, AT&T Labs Research Technical Report TD-6XLTAY, Florham Park, NJ, USA.

The authors derive equations for the point and line correspondences, explicitly accounting for noise in the image data, and provide a mixed integer nonlinear optimization formulation for the object recognition problem. They also propose a decomposition approach to solve it and describe details of a C-GRASP, that is applied in the first step of the decomposition technique for the scenarios considered in the computational study. RCL is built by applying the min-max α-percentage rule. approximating the role of the gradient of the objective function.

.

Hirsch, M., P. Pardalos, and M. Resende (2006b). Speeding up continuous grasp. Technical report, AT&T Labs Research Technical Report TD-6U2P2H, Florham Park, NJ, USA.

The authors propose some improvements to speed up the original continuos GRASP (C-GRASP) and to make it more robust. The main differences between this new version of C-GRASP and the previous one do not involve construction and local search mechanisms, but are limited only to the input parameters and when the grid density is increased. The authors compare the new C-GRASP with the original version as well as with other algorithms from the literature on a set of benchmark multimodal test functions whose global minima are known.

.

Jr., H. F., S. B. M. Resende, and D. Falco (2005). Power transmission network design by a greedy randomized adaptive path relinking approach. *IEEE Transactions on Power Systems* **20**(1), 43–49.

This paper proposes a new metaheuristic approach called Greedy Randomized Adaptive Path Relinking (GRAPR), to solve the static power transmission network design problem, which consists in choosing, from a pre-defined set of candidate circuits, those that should be built in order to minimize the investment and operational costs, and to supply the forecasted demand along a planning horizon. GRAPR uses in the path relinking phase generalized GRASP concepts to explore different trajectories between two good quality solutions previously found, such as the RCL idea of selecting not only the best ranked candidate, but a set of well ranked candidates. The authors describe computational results obtained from two real-world case studies of Brazilian systems.

.

Laguna, M. and R. Martí (2001). A grasp for coloring sparse graphs. *Computational Optimization and Applications* **19**(2), 165–178.

During GRASP construction phase the greedy criterion is defined on vertices degree. The local search procedure combines into one the smallest and second

smallest cardinality color classes and then, for each violating vertex it tries to rebuild feasibility.

.

Lim, A., B. Rodrigues, and C. Wang (2006). Two-machine flow shop problems with a single server. *Journal of Scheduling* **9**, 515–543.

Several heuristics are proposed in this paper to solve the problems in general case, including simulated annealing, TS, genetic algorithms, GRASP, and other hybrids. In the construction phase two greedy criteria are adopted. One criterion is to sort by setup times instead of the processing times. The second criterion schedules all jobs one by one and each insertion will increase the objcctive function by a minimum value. For the local search phase, the authors define several neighborhoods, including swapping, insertion, and a mixture of them.

.

Lim, A. and F. Wang (2004). A smoothed dynamic tabu search embedded GRASP for *m*-VRPTW. In *Proceedings of ICTAI 2004*, pp. 704–708.

m-VRPTW is a vehicle routing problem with both time window and limited number of vehicles (m-VRPTW). In this paper an improved GRASP is proposed that uses techniques including multiple initialization and solution reuse. A smoothed dynamic tabu search is also embedded into the GRASP to improve the performance.

.

Loureno, H. and D. Serra (2002). Adaptive approach heuristics for the generalized assignment problem. *Mathware and Soft Computing* **9**(2-3), 209–234.

This paper presents several hybrid algorithms consisting of adaptive construction heuristics and subsequent application of local search to solve the GAP. The basic elements are extract from a specific Ant Colony Optimization algorithm (MAX-MIN Ant System) and GRASP, which are are embedded in a general framework characterized by three steps. In the first step, a solution is constructed by a randomized construction heuristic; in the second step, a local search is applied to improve the just built solution, a descent local search and TS are proposed; the last step consists in updating a set of parameters which guide the construction process. The three steps are repeated until a stopping criterion is verified. The choices made in each step lead to different heuristic methods. The main difference between GRASP and the Ant Colony Optimization algorithm is the method of selecting the agent to whom the previously chosen task is assigned. For GRASP the choice is a probabilistic bias according to an adaptive priority function, which does not depend on the solutions seen in previous iterations of the general framework but only on the choices done in the previous iterations of GRASP. the local search procedure interchanges or reassigns tasks. The authors admit unfeasible solutions with respect to capacity constraints, i.e. the total resource required by the tasks assigned to some agents may exceed their capacity.

.

Martí, R. and M. Laguna (2003). Heuristics and meta-heuristics for 2-layer straight line crossing minimization. *Discrete Applied Mathematics* **127**(3), 665–678.

This paper presents extensive computational experiments to compare several different randomized metaheuristics for the problem of minimizing straight-line crossings in a 2-layer graph. A TS metaheuristic yields the best results for relatively dense graphs, while a GRASP outperforms all other approaches when tackling low-density graphs. The GRASp construction phase starts by creating a list of unassigned vertices, originally consists of all the vertices in the graph. The greedy criterion takes into account vertices degree. At each iteration of the local search phase vertices are selected to be moved and the probability of selection if a vertex increases with its degree. When a vertex is selected, three moves are considered: (1) to insert the vertex one position above the barycenter, (2) to insert the vertex at the barycenter position, and (3) to insert the vertex one position below the barycenter. The vertex is placed in the position that produces the maximum reduction in the number of crosses.

.

Oliveira, C., P. Pardalos, and M. Resende (2004). GRASP with path-relinking for the quadratic assignment problem. In *Efficient and Experimental Algorithms*, Volume 3059 of *Lecture Notes in Computer Science*, pp. 356–368.

A GRASP for the quadratic assignment problem is described. Construction first makes two assignments, and then completes the solution by making assignments, one at a time. The greedy function is assignment interaction cost. The local search procedure is a 2 assignment exchange. Path relinking is invoked at each GRASP iteration as intensification procedure.

.

Osman, I., B. Al-Ayoubi, and M. Barake (2003). A greedy random adaptive search procedure for the weighted maximal planar graph problem. *Computers and Industrial Engineering* **45**(4), 635–651.

The weighted maximal planar graph (WMPG) problem seeks to find a subgraph from a given weighted complete graph such that the subgraph is planar–it can be embedded on the plane without any arcs intersecting. The subgraph is maximal no additional arc can be added to the subgraph without destroying its planarity and it also has the maximal sum of arc weights. In this paper, the authors describes a GRASP, where RCL is dynamically updated.

.

Pacheco, J. and S. Casado (2005). Solving two location models with few facilities by using a hybrid heuristic: a real health resources case. *Computers and Operations Research* **32**, 3075–3091.

A hybrid scatter search algorithm is proposed that incorporates procedures based on different strategies, such as GRASP and path relinking. GRASP is used as diversification method and the greedy criterion is to take into account the value of the objective function if a certain location is added to the solution. Local search is based on interchange neighborhood.

.

Pitsoulis, L. and M. Resende (2002). Greedy randomized adaptive search procedures. In P. Pardalos and M. Resende (Eds.), *Handbook of Applied Optimization*, pp. 168–183. Oxford University Press.

This chapter gives an overview of GRASP. Besides describing the basic building blocks of a GRASP, it covers enhancements to the basic procedure, including reactive GRASP, hybrid GRASP, and intensification strategies.

.

Piana, E., I. Plana, V. Campos, and R. .Martí (2004). .

This paper proposes a GRASP for the problem of reducing the bandwidth of a matrix, i.e. of finding a permutation of the rows and columns of a given matrix, which keeps the nonzero elements in a band that is as close as possible to the main diagonal. The proposed GRASP is combined with a Path Relinking strategy as intensification to search for improved outcomes. Given a matrix $A = \{a_{ij}\}_{n\times n}$, the problem can be stated as graph theory problem considering a vertex for each row and a vertex for each column. Then, an edge (i, j) is inserted if either $a_{ij} \neq 0$ or $a_{ji} \neq 0$ and the problem consists of finding a labeling f of the vertices that minimizes the maximum difference between labels of adjacent vertices. The authors propose and test five different GRASP constructive methods. The first method C_1 starts by selecting at random a vertex and assigning to it a random label. Then, at each iteration the candidate elements CL are all vertices that are adjacent to at least one labeled vertex and RCL is formed from those candidate vertices with a maximum number of adjacent labeled vertices. Constructive methods C_2 and C_3 differ from C_1 in the definition of the RCL. At each iteration, in C_2 RCL elements are those vertices that have been in CL for a maximum number of construction steps, while C_3 combines both criteria by considering the attractiveness of a vertex as the sum of both measures: the number of adjacent labeled vertices and the number of steps that a vertex has been in CL. Constructive methods C_4 and C_5 are partially based on the construction of a level structure of vertices set proposed in the GPS method, i.e. a special partition of the set of vertices. The local search considers the changes in the number of critical vertices before and after a move of a vertex.

.

Resende, M. and C. Ribeiro (2002). Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger (Eds.), *State-of-the-Art Handbook in Metaheuristics*. Kluwer Academic Publishers.

This chapter gives an overview of GRASP. Besides describing the basic building blocks of a GRASP, it covers enhancements to the basic procedure, including reactive GRASP, hybrid GRASP, and intensification strategies. Applications are also discussed.

.

Resende, M. and C. Ribeiro (2003). A grasp with path-relinking for private virtual circuit routing. *Networks* **41**, 104–114.

In this paper, the offline Private Virtual Circuit Routing (PVC) problem is formulated as an integer multicommodity flow problem with additional constraints and with an objective function that minimizes propagation delays and/or network congestion. The authors of this paper propose variants of a GRASP with path relinking heuristic for this problem. In the construction phase, the routes are determined, one at a time. At each iteratin, RCL is built by applying the

fixed-size rule and it is formed by a certain fixed number of unrouted PVC pairs with the largest demands. From RCL a pair is then selected with a probability computed trough a strategy usually employed by heuristic-biased stochastic sampling. Once a PVC is selected, it is routed on a shortest path from its origin to its destination. Each solution built in the construction phase may be viewed as a set of routes, one for each PVC. the local search procedure seeks to improve each route in the current solution, removing some units of flow from each edge in its current route and trying to route.

.

Resende, M. and C. Ribeiro (2005). Parallel greedy randomized adaptive search procedures. In E. Alba (Ed.), *Parallel Metaheuristics: A new class of algorithms*, pp. 315–346. John Wiley and Sons.

In this chapter, the authors survey parallel implementations of GRASP. They describe simple strategies to implement independent parallel GRASP heuristics and more complex cooperative schemes using a pool of elite solutions to intensify the search process. Some applications of independent and cooperative parallelizations are presented in detail.

.

Resende, M. and J. G. Velarde (2003). Grasp: Procedimientos de bsqueda miope aleatorizado y adaptatitvo. *Inteligencia Artificial* (19), 61–76.

In this chapter, the authors describe the basic components of GRASP as well as successful implementation techniques and parameter tuning strategies. In Spanish.

.

Ribeiro, C. and I. Rosseti (2002). A Parallel GRASP Heuristic for the 2-Path Network Design Problem. In *Proceedings of Euro-Par 2002*, Volume 2400 of *Lecture Notes in Computer Science*, pp. 922–926.

In this article, the authors propose a parallel cooperative strategy for GRASP with path relinking for the 2-path network design problem. At each construction phase iteration a pair of nodes is selected at random and a shortest path connecting them is computed. Neighbors of a solution S are obtained by replacing in S any of its 2-paths by another 2-path between the same origin-destination pair.

.

Ribeiro, C. and I. Rosseti (2007). Efficient parallel cooperative implementations of grasp heuristics. *Parallel Computing* **33**, 21–35.

In this article, the authors propose and analyze parallel cooperative strategies for GRASP with path relinking for the 2-path network design problem. At each construction phase iteration a pair of nodes is selected at random and a shortest path connecting them is computed. Neighbors of a solution S are obtained by replacing in S any of its 2-paths by another 2-path between the same origin-destination pair. pargrasp.pdf.

.

Rocha, P., M. Ravetti, and G. Mateus (2004). The metaherusitic grasp as an upper bound for a branch and bound algorithm in a scheduling problem with non-related parallel machines and sequence-dependent setup times. In *Proceedings of the 4th EU/ME Workshop: Design and Evaluation of Advanced Hybrid Meta-Heuristica*, Volume 1, pp. 62–67.

A scheduling problem with sequence-dependent setup time is among the most complex scheduling problems. In fact, considering one machine and one stage, the problem is equivalent to the TSP. While there is much research work considering parallel machines, few papers consider parallel non-related machines and sequence-dependent setup times and all of them are randomized local serch based heuristics. In this paper the authors propose an hybridization of GRASP as an upper bound for a Branch and Bound procedure. At each GRASP iteration the greedy criterion adopted to build a feasible solution consists in sorting the jobs in a non-decreasing order using the due date and then assigning each job to the machine able to finishing it first. The local search switches all existing pairs of jobs assigned to different machines. The authors have also implemented path relinking (PR) at the end of each GRASP iteration to intensify the local search. First, a pool of good solutions (Elite Set) is retained. Every time PR is used, a solution is randomly chosen from this pool. Then, all solutions in the path from the solution found in the local search to the selected solution from the pool are spanned. If a better solution is found, it is added to the pool.

.

Souza, M., C. Duhamel, and C. Ribeiro (2004). A grasp heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy. In M. Resende and J. Souza (Eds.), *Metaheuristics: Computer Decision-Making*, pp. 627–658. Kluwer Academic Publisher.

The authors propose a GRASP with path-relinking heuristic. It uses a randomized version of a savings heuristic in the construction phase and an extension of the local search strategy, incorporating some short term memory elements of tabu search. The greedy criterion takes into account the saving with respect to edge costs and capacity constraints. The local search strategy is based on a new neighborhood structure defined by path exchanges.

.