Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 153 (2004) 564–580
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

GRASP for set packing problems

Xavier Delorme [a,b,*], Xavier Gandibleux [b], Joaquin Rodriguez [a]

[a] *INRETS-ESTAS, 20 rue Elisée Reclus, F-59650 Villeneuve d'Ascq, France*
[b] *LAMIH—UMR CNRS 8530, Université de Valenciennes, Campus "Le Mont Houy", F-59313 Valenciennes Cedex 9, France*

Received 10 December 2001; accepted 30 January 2003

Abstract

The principles of the Greedy Randomized Adaptative Search Procedure (GRASP) metaheuristic are instantiated for the set packing problem. We investigated several construction phases, and evaluated improvements based on advanced strategies. These improvements include a self-tuning procedure (using reactive GRASP), an intensification procedure (using path relinking) and a procedure involving the diversification of the selection (using a learning process). Two sets of various numerical instances were used to perform the computational experiments. The first set contains randomly generated instances, while the second includes instances relating to real problems in railway planning. No metaheuristic has previously been applied to this combinatorial problem. Consequently, we have discussed GRASP's performances both in relation to lower/upper bounds and to the results obtained with Cplex when such results are available. Our analysis, based on the average performances observed, shows the impact of the suggested strategies, and indicates the configuration that produces the best results.

Keywords: Combinatorial optimization; Set packing problem; Metaheuristic; Reactive GRASP; Path relinking; Railway planning problem

1. Introduction

This study concerns the resolution of the "set packing problem" (SPP), a classical optimization problem, close to the "set covering problem" [1,2], which can also be reformulated as a "node packing problem". Surprisingly, the SPP has not received much attention in the literature. In comparison with the "set covering" and "node packing" problems, very few studies have examined the SPP resolution.

The impetus for our research comes from a real railway problem which can be formulated as an SPP [3,4]. The principal concerns revolve around the evaluation of infrastructure capacity in a railway network. The numerical instances used in this study come from the Pierrefitte Gonesse junction, north of Paris. However, because these instances have a specific structure, they do not allow us to obtain general

* Corresponding author. Address: INRETS-ESTAS, 20 rue Elisée Reclus, F-59650 Villeneuve d'Ascq, France.
E-mail addresses: xavier.delorme@inrets.fr (X. Delorme), xavier.gandibleux@univ-valenciennes.fr (X. Gandibleux), joaquin.rodriguez@inrets.fr (J. Rodriguez).
URLs: http://www3.inrets.fr/~delorme/, http://www.univ-valenciennes.fr/ROAD/XavierG/.

0377-2217/$ - see front matter
doi:10.1016/S0377-2217(03)00263-7

information concerning the advantages and disadvantages of a resolution method. For this reason, we have also used several randomly generated instances of a similar size, but with more varied characteristics.

In the first stage of our study, we used Cplex to resolve the problem [5]. However, given the size of the numerical instances, exact resolution was impossible within a reasonable period of time. Our second stage involved the development of a heuristic method derived from the GRASP (Greedy Randomized Adaptative Search Procedure) metaheuristic [6]. Several general algorithms, which can solve any SPP, were developed. Within the basic GRASP framework, we considered three different greedy methods, including one involving a new learning process, as well as two extensions of this standard framework (reactive GRASP and path relinking). We compared their abilities to solve a range of SPP, particularly those where Cplex cannot produce good quality results. This comparison demonstrates the efficiency of the various parameters and strategies with regard to the characteristics of the instances. This information will be useful in future studies when we will seek to develop a particular algorithm for our specific railway problem.

Section 2 introduces the SPP and its particularities. The principles of GRASP are reviewed in Section 3, along with the algorithmic description of our different operational procedures. In Sections 4 and 5, we report and comment on the characteristics of our instances (including a description of the railway problem) (Section 4) and the numerical results collected (Section 5). Section 6 concludes the discussion.

2. The set packing problem

Given a finite set $I = \{1, \ldots, n\}$ of valuated items and $\{T_j\}, j \in J = \{1, \ldots, m\}$, a collection of subsets of I, a packing is a subset $P \subseteq I$ such that $|T_j \cap P| \leqslant 1, \forall j \in J$. The set J can also be seen as a set of constraints between the items of the set I. The objective of the SPP is to maximise the total value of the packing obtained. This problem can be formulated with a mathematical model (1):

$$\left[\begin{array}{lll} \text{Max } z = \sum_{i\in I} c_i x_i, & & \\ \sum_{i\in I} t_{i,j} x_i \leqslant 1, & \forall j \in J, & \\ x_i \in \{0,1\}, & \forall i \in I, & \\ t_{i,j} \in \{0,1\}, & \forall i \in I, \forall j \in J & \end{array}\right] \tag{1}$$

considering

- a vector $x = (x_i)$ where $x_i = \begin{cases} 1 & \text{if } i \in P \\ 0 & \text{otherwise} \end{cases}$,
- a vector $c = (c_i)$ where $c_i =$ value of the item i,
- a matrix $t = (t_{i,j})$ where $t_{i,j} = \begin{cases} 1 & \text{if } i \in T_j \\ 0 & \text{otherwise} \end{cases}$.

The node packing problem (2) is a specific case of the SPP in which the constraints are between pairs of items:

$$\left[\begin{array}{lll} \text{Max } z = \sum_{i\in I} c_i x_i, & & \\ x_i + x_j \leqslant 1, & \forall \text{ pair } (i,j) \text{ of incompatible items}, & \\ x_i \in \{0,1\}, & \forall i \in I & \end{array}\right]. \tag{2}$$

Every SPP can be formulated as a node packing problem, where each constraint is split into several constraints, one for each pair of items. However, this reformulation increases the number of constraints; each constraint involving n items is rewritten as C_n^2 constraints on two items. Every node packing problem can also be expressed as a set covering problem with a simple change of variable ($x'_i = 1 - x_i, \forall i \in I$).

The SPP is known to be strongly NP-Hard, according to Garey and Johnson [7]. Its particularities are described in more detail by Gondran and Minoux [1] and Nemhauser and Wolsey [2]. The best exact method known for solving this problem is a Branch & Cut algorithm using polyhedral theory to obtain

facets, essentially by determining cliques as defined by Padberg [8]. However, only small-sized instances can be solved exactly. To our knowledge, and according to Osman and Laporte [9], no one has ever applied metaheuristics to SPP resolution.

Surprisingly, few applications of the SPP formulation have been reported in the literature. Of those, five recent applications are summarized briefly in chronological order in the lines below. Rönnqvist [10] worked on a cutting stock problem formulated as an SPP and solved using a Lagrangian relaxation combined with subgradient optimization. Zwaneveld et al. [11] formulated a real railway feasibility problem as an SPP and solved it exactly using reduction tests and a Branch & Cut method. Kim [12] represented a ship scheduling problem as an SPP and used LINDO software to solve it. Mingozzi et al. [13] used an SPP formulation to calculate the bounds for a Resource Constrained Project Scheduling Problem using a greedy method. Rossi [14] considered an SPP formulation for a ground holding problem and solved it exactly with a Branch & Cut method.

3. Algorithmic description of the operational procedures

3.1. Overview of GRASP

GRASP is a multistart two-phase metaheuristic for combinatorial optimization proposed by Feo and Resende [6]. The first phase is a construction phase that builds an initial solution using a greedy randomized procedure, whose randomness allows solutions in different areas of the solution space to be obtained. The second phase is a local search phase that improves these solutions. This two-phase process is reiterative (for a general overview of GRASP, see the article by Pitsoulis and Resende [15]). Several new components have extended the scheme of GRASP (reactive GRASP, parameter variations, bias functions, memory and learning, improved local search, path relinking, hybrids,...). These components are presented and discussed in Resende and Ribeiro [16].

To implement GRASP for a specific problem, six main choices must be made:

- the greedy method,
- the random character importance, fixed by a parameter $\alpha \in [0;1]$,
- the neighbourhood considered for the local search,
- the possible intensification phase,
- the stopping criterion,
- the possible post-optimization phase.

The complete GRASP scheme is described in the Algorithm 1, in which the optional phases are indicated in brackets.

Algorithm 1. The GRASP algorithm

$Solutions \leftarrow \emptyset$
repeat
  $initialSol \leftarrow greedyRandomized(problem, \alpha)$
  $improvedSol \leftarrow localSearch(initialSol)$
  $[improvedSol \leftarrow intensification(improvedSol)]$
  $Solutions \leftarrow Solutions \cup \{improvedSol\}$
until *stopping criteria*
$[Solutions \leftarrow postOptimization(Solutions)]$
$finalSol \leftarrow best(Solutions)$

It is easy to customize the GRASP metaheuristic to solve any problem for which construction and local search algorithms are available. GRASP has been applied to a wide range of optimization problems, including academic and industrial problems in scheduling, routing, logic, partitioning, location and layout, graph theory, assignment, manufacturing, transportation, telecommunications, electrical power systems, and VLSI design. An extensive annotated bibliography has been proposed by Festa and Resende [17]. Several studies have shown that GRASP produces good quality solutions for hard combinatorial optimization problems, particularly the set covering problems [18–21], the node packing problems [19,22] and the set packing problems [3,4].

3.2. The greedy randomized phase

For the greedy phase of our GRASP algorithm, we propose three different greedy algorithms (called greedy1, greedy2 and greedy3). Their characteristics are described in the following sections, respectively 3.2.1, 3.2.2 and 3.2.3.

3.2.1. The greedy1 algorithm

The greedy1 procedure (Algorithm 2) is inspired by some of the implementations [18,20,21] of a GRASP for the set covering problem proposed by Féo and Resende [19]. This procedure builds a solution from the trivial non-feasible solution, $x_i = 1, \forall i \in I$. Several variable values are flipped to 0 until a feasible solution is obtained. These changes concern only one variable per iteration. To keep a maximal value for the objective function, variables involving a maximum number of constraints with a minimum value (i.e. with c_i minimum) are prioritized. A Restricted Candidate List (RCL) is constituted of the variables at the top of the priority listing according to a threshold parameter $\alpha \in [0,1]$ applied to their evaluation. The choice of the variable fixed to 0 is random among the variables included in the RCL. With a value $\alpha = 0$, the algorithm corresponds to a random construction; with a value $\alpha = 1$, it is equivalent to a greedy algorithm.

Algorithm 2. The greedy randomized construction algorithm greedy1

$I_t \leftarrow I$
$x_i \leftarrow 1, \forall i \in I_t$
$J_t \leftarrow J \setminus \{j : \sum_{i \in I_t} t_{i,j} x_i \leqslant 1\}$
while $(J_t \neq \emptyset)$ loop
  $Eval_i \leftarrow \sum_{j \in J_t} t_{i,j}/c_i, \forall i \in I_t$
  $Limit \leftarrow \min_{i \in I_t}(Eval_i) + \alpha * (\max_{i \in I_t}(Eval_i) - \min_{i \in I_t}(Eval_i))$
  $RCL \leftarrow \{i \in I_t, Eval_i \geqslant Limit\}$
  $i^* \leftarrow RandomSelect(RCL)$
  $x_{i^*} \leftarrow 0$
  $I_t \leftarrow I_t \setminus \{i^*\}$
  $J_t \leftarrow J_t \setminus \{j : \sum_{i \in I_t} t_{i,j} x_i \leqslant 1\}$
endWhile

Considering n as the number of variables and m as the number of constraints, the complexity of the greedy1 algorithm is $O(n^2 m)$ in the worst case.

3.2.2. The greedy2 algorithm

The greedy2 procedure (Algorithm 3) is inspired by a GRASP for the node packing problem proposed by Féo et al. [19,22]. This procedure builds a solution from the trivial feasible solution, $x_i = 0, \forall i \in I$. Some variable values are set to 1, as long as the solution is maintained feasible. Changes concern only one

variable at each iteration. To increase the objective function, variables which involve a minimum number of constraints with a maximum value are prioritized, but the choice is random among the most interesting variables. Changes stop when no variable can be fixed to 1 without losing feasibility.

Algorithm 3. The greedy randomized construction algorithm greedy2

$I_t \leftarrow I$
$x_i \leftarrow 0, \forall i \in I_t$
$Eval_i \leftarrow c_i / \sum_{j \in J} t_{i,j}, \forall i \in I_t$
while $(I_t \neq \emptyset)$ loop
  $Limit \leftarrow \min_{i \in I_t}(Eval_i) + \alpha * (\max_{i \in I_t}(Eval_i) - \min_{i \in I_t}(Eval_i))$
  $RCL \leftarrow \{i \in I_t, Eval_i \geqslant Limit\}$
  $i^* \leftarrow RandomSelect(RCL)$
  $x_{i^*} \leftarrow 1$
  $I_t \leftarrow I_t \setminus \{i^*\}$
  $I_t \leftarrow I_t \setminus \{i : \exists j \in J, t_{i,j} + t_{i^*,j} > 1\}$
endWhile

The complexity of the greedy2 algorithm is $O(\beta nm)$ in the worst case, with β representing the maximum number of iterations. This number is equal to the maximal number of variables that can be fixed to 1 in a feasible solution. This means that the complexity can theoretically be $O(n^2 m)$ (where $\beta \leqslant n$), but in practice β is often very small compared to n. Thus, the complexity of the greedy2 algorithm is less than that of the greedy1 algorithm (Section 3.2.1).

3.2.3. The greedy3 algorithm

The greedy3 procedure (Algorithm 4) is an evolution of the greedy2 algorithm described in Section 3.2.2 and includes a learning process. It seeks to improve on the evaluation of the variables throughout the main loop of the GRASP algorithm. Each constraint is weighted by its frequency saturation on the generated set of solutions. In the Algorithm 4, *SaturationFrequency*(j) denotes the ratio of the number of times the constraint j has been saturated to the number of generated solutions. A constraint is considered saturated by a solution if a variable appearing in this constraint is fixed to 1 in this solution (i.e. $\exists i \in I$, $x_i = 1$ and $t_{i,j} = 1$). This computation is done by a function *Update*( ). *SaturationFrequency* is initialized to 0 for the first GRASP iteration.

Algorithm 4. The greedy randomized construction algorithm greedy3

$I_t \leftarrow I$
$x_i \leftarrow 0, \forall i \in I_t$
$Eval_i \leftarrow c_i / \sum_{j \in J}(t_{i,j} * (1 + SaturationFrequency(j))), \forall i \in I_t$
while $(I_t \neq \emptyset)$ loop
  $Limit \leftarrow \min_{i \in I_t}(Eval_i) + \alpha * (\max_{i \in I_t}(Eval_i) - \min_{i \in I_t}(Eval_i))$
  $RCL \leftarrow \{i \in I_t, Eval_i \geqslant Limit\}$
  $i^* \leftarrow RandomSelect(RCL)$
  $x_{i^*} \leftarrow 1$
  $I_t \leftarrow I_t \setminus \{i^*\}$
  $I_t \leftarrow I_t \setminus \{i : \exists j \in J, t_{i,j} + t_{i^*,j} > 1\}$
endWhile
$Update(SaturationFrequency(j)), \forall j \in J$

As with the greedy2 algorithm, the complexity of this algorithm is $O(\beta nm)$.

3.3. The choice of the α parameter: reactive GRASP

Preliminary experimentation showed us that no particular value for the α parameter can be determined as best for all or even a large part of the instances. The same observation was reported by Resende and Ribeiro [16]. As a result, we decided to consider several values for the α parameter. Our first strategy was to choose α randomly from a set *alphaSet* with a uniform discrete probability distribution. However, we also examined a second, reportedly better, strategy called reactive GRASP, in which the parameter α is self-adjusted according to the quality of the solutions previously obtained. The algorithm that we propose (Algorithm 5) is an evolution of Algorithm 1 which is inspired by the GRASP algorithm of Prais and Ribeiro [23].

Starting from a uniform discrete probability distribution among each value of the set *alphaSet*, $proba_\alpha$ is updated periodically according to a *probaUpdate* condition. New probabilities are calculated from the average value of the elite solutions obtained for each α value (the elite solutions are listed in a $Pool_\alpha$ with a limited size). The parameter δ is introduced to attenuate the updated values of the probabilities p_i.

Algorithm 5. The reactive GRASP algorithm

$Solutions \leftarrow \emptyset$
$proba_\alpha \leftarrow 1/|alphaSet|, \forall \alpha \in alphaSet$
repeat
  $\alpha^* \leftarrow RandomSelect(alphaSet, proba)$
  $initialSol \leftarrow greedyRandomized(problem, \alpha^*)$
  $improvedSol \leftarrow localSearch(initialSol)$
  $[improvedSol \leftarrow intensification(improvedSol)]$
  $Solutions \leftarrow Solutions \cup \{improvedSol\}$
  $Pool_{\alpha^*} \leftarrow Pool_{\alpha^*} \cup \{improvedSol\}$
  if *probaUpdate condition is true* then

$$valuation_\alpha \leftarrow \left(\frac{mean_{s \in Pool_\alpha}(z(s)) - z(worst(Solutions))}{z(best(Solutions)) - z(worst(Solutions))} \right)^\delta, \forall \alpha \in alphaSet$$

$$proba_\alpha \leftarrow valuation_\alpha \Big/ \left(\sum_{\alpha' \in alphaSet} valuation_{\alpha'} \right), \forall \alpha \in alphaSet$$

  endIf
until *stopping criteria*
$[Solutions \leftarrow postOptimization(Solutions)]$
$finalSol \leftarrow best(Solutions)$

3.4. The local search phase

The neighbourhood $\mathcal{N}$ used for the local search procedure (Algorithm 6) is based on k–p exchanges. The k–p exchange neighbourhood of a solution x is the set of solutions obtained from x by changing the value of k variables from 1 to 0, and changing p variables from 0 to 1. Due to the combinatorial explosion of the number of exchange possibilities when k and p increase, we were obliged to limit them, testing only 0–1 exchanges, 1–1 exchanges, 2–1 exchanges and 1–2 exchanges. Moreover, the search procedure was implemented using a first-improving strategy (i.e. we selected the first neighbour whose value is better than the

current solution). Whenever an exchange is accepted, the local search is re-started from this new solution. The local search stops when no further improving exchanges are possible.

The complexity of the *searchNeighbourhood* algorithms (for 1–2 exchanges and 2–1 exchanges) is $O(n^4m)$ in the worst case. However, the practical time can be significantly reduced when the local search algorithm is applied to a good initial solution.

Algorithm 6. The local search algorithm

```
function searchNeighbourhood(s, 𝒩)
   while s not locally optimal loop
      Find s' ∈ 𝒩(s) with z(s') > z(s)
      s ← s'
   endWhile
   return s
end searchNeighbourhood
repeat
   Solution ← searchNeighbourhood(Solution, 0–1exchanges)
   Solution ← searchNeighbourhood(Solution, 1–1exchanges)
   Solution ← searchNeighbourhood(Solution, 2–1exchanges)
   Solution ← searchNeighbourhood(Solution, 1–2exchanges)
until Solution not improved
return Solution
```

3.5. The intensification phase: path relinking

The intensification method is based on path relinking, which was originally proposed for the tabu search by Glover and Laguna [24]. With this procedure, paths from one elite solution to another are generated (in the solution space) and explored to obtain better solutions. Path relinking was first used as an intensification of a GRASP procedure by Laguna and Martí [25]. However, our path relinking algorithm for the SPP (Algorithm 7) is inspired by the one proposed by Resende and Ribeiro [26].

In the intensification method, the best solutions obtained by GRASP are placed in a *Pool* with a limited size. For each GRASP iteration, the path relinking algorithm is applied to the solution generated by the local search phase and to one solution randomly selected among those in the *Pool*. The best of these two solutions is taken as the *initialSolution* and the other as the *targetSolution*. At each iteration of the path relinking, the solution is repaired if necessary. The *repairing* function used is the greedy1 algorithm described in Section 3.2.1, with a value of 1 for the α parameter (i.e. a pure greedy procedure). Then the solution is saturated, using a *saturation* function which is the greedy2 algorithm described in Section 3.2.2, also with a value of 1 for α. The solutions thus generated can also be included in the *Pool*.

Algorithm 7. The path relinking algorithm

```
Solution ← initialSolution
for i ∈ I loop
   if initialSolution(i) ≠ targetSolution(i) then
      Solution(i) ← targetSolution(i)
      currentSolution ← Solution
      if currentSolution not feasible then
         currentSolution ← repairing(currentSolution)
      endIf
```

$currentSolution \leftarrow saturation(currentSolution)$
$Pool \leftarrow Pool \cup \{currentSolution\}$
endIf
endFor

Since the complexity of the *repairing* algorithm is $O(n^2m)$, the complexity of this algorithm is $O(n^3m)$ in the worst case.

4. Numerical instances

Given the lack of available SPP benchmarks, we decided to consider two types of instances to evaluate our GRASP implementations. The first type consists of randomly generated instances, and the second type is related to a real railway problem.

4.1. Randomly generated instances

Several instances were generated randomly using the following parameters:

- The number of variables is equal to 1,000 or 2,000.
- The number of constraints is equal to 100% or 500% of the number of variables.
- The maximum number of non-null elements by constraint is equal to 1% or 5% of the number of variables (i.e. the actual number of non-null elements is a random number between two and this parameter).
- The values of items (c_i) are uniformly distributed in the interval [1–20] (weighted) or fixed to 1 (unicost).

All combinations of these parameters were considered. Sixteen different instances, labeled Rnd, were thus generated.

The characteristics of these instances are presented in Table 1. The density corresponds to the percentage of non-null elements in the constraint matrix. We also indicated the weighted instances. The objective value

Table 1
Characteristics of the randomly generated instances

No.	Initial problem					Best known	Reduced problem	
	$\|I\|$	$\|J\|$	Density	c_i	LP		$\|I\|$	Reduction
1	1,000	5,000	2.60%	[1–20]	330.26	67*	997	0.3%
2	1,000	5,000	2.59%	[1–1]	22.81	4*	999	0.1%
3	1,000	5,000	0.60%	[1–20]	1365.64	661	1,000	0.0%
4	1,000	5,000	0.60%	[1–1]	111.77	48	1,000	0.0%
5	1,000	1,000	2.60%	[1–20]	434.71	222	1,000	0.0%
6	1,000	1,000	2.65%	[1–1]	29.91	15	1,000	0.0%
7	1,000	1,000	0.58%	[1–20]	2349.19	2,255	949	5.1%
8	1,000	1,000	0.60%	[1–1]	184.32	175	920	8.0%
9	2,000	10,000	2.54%	[1–20]	339.62	40*	513	74.4%
10	2,000	10,000	2.54%	[1–1]	22.59	2*	476	76.2%
11	2,000	10,000	0.55%	[1–20]	1500.10	478	2,000	0.0%
12	2,000	10,000	0.55%	[1–1]	114.01	32	2,000	0.0%
13	2,000	2,000	2.55%	[1–20]	423.98	140	2,000	0.0%
14	2,000	2,000	2.56%	[1–1]	28.70	9	2,000	0.0%
15	2,000	2,000	0.56%	[1–20]	2209.57	1,784	2,000	0.0%
16	2,000	2,000	0.56%	[1–1]	168.82	131	2,000	0.0%

of the linear relaxation (LP) is also shown, as well as that of the best known solution, with an asterisk if it is optimal (note that there can be quite a significant gap between the values for the best known solutions and the linear relaxation). In addition, the number of non-redundant variables is given, as well as the percentage of reduction that can be obtained by removing the redundant variables. This number was obtained by applying a dominance test algorithm (Algorithm 8).

Algorithm 8. The dominance test algorithm

```
for i1 ∈ I loop
  if {i2 ∈ I \ {i1}, (∀j ∈ J, t_{i1,j} ≥ t_{i2,j}) and (c(i1) ≤ c(i2))} ≠ ∅ then
    I ← I \ {i1}
    J ← J \ {j ∈ J, t_{i1,j} = 1 and Σ_{i∈I} t_{i,j} ≤ 2}
  endIf
endFor
```

The dominance test algorithm compares each pair of variables. We consider that a variable i_1 is dominated by another variable i_2, if i_1 saturates all the constraints that i_2 saturates (and potentially more) and if the value of the item i_1 is less than or equal to that of the item i_2. Such a variable i_1 can be removed from the problem without changing its optimal solution. Additionally, constraints which have only one $t_{i,j}$ different of zero after removing the variables can also be removed from the problem.

4.2. Railway problem instances

The second set of SPP instances considered in the evaluation of our GRASP implementations comes from an actual railway planning problem. Railway infrastructure managers now have to deal with operators' requests for increased capacity. Planning the construction or reconstruction of infrastructures must be done very carefully in light of the huge investments required. Usually, assessing the capacity of one component of a rail system is done by measuring the maximum number of trains that can be operated on this component within a certain time period. Measuring the capacity of junctions is a matter of solving an optimization problem called the *feasibility problem*. As mentioned in Section 2, this problem has been formulated in terms of an SPP by Zwaneveld et al. [11] and more recently by Delorme et al. [3,4]. The feasibility problem of a junction can be stated as follows:

Given the layout of a junction and a set of trains $\mathscr{T}$, *how many trains from* $\mathscr{T}$ *can be routed through the junction within a certain time period such that all safety constraints are satisfied*?

The construction of the set $\mathscr{T}$ is detailed in Delorme et al. [4]. Each train of the set $\mathscr{T}$ has an allowed set of routes and an allowed set of entrance times. To design the feasibility problem, binary decision variables are introduced. These decision variables are

$$x_i = \begin{cases} 1 & \text{if the combination } i \text{ of a train, a route and an entrance time is used,} \\ 0 & \text{otherwise.} \end{cases}$$

This formulation is close to the model developed in Zwaneveld et al. [11], in which the decisions of the entrance times were not included. The objective is to maximize the number of trains that can be routed through the junction without preference between trains. This is reflected in the objective function with the item coefficients $c_i = 1, \forall i \in I$. This particularity raises a unicost SPP (USPP).

The set I of items corresponds to the set of the allowed combinations of trains/routes/entrance times; the set J corresponds to the set of constraints between these combinations. There are two types of constraints. The first ensures that only one route and one entrance time value can be set for each train. The second

Table 2
Characteristics of the railway instances

No.	Initial problem					Best known	Reduced problem	
	n_S	$\|I\|$	$\|J\|$	Density	LP		$\|I\|$	Reduction
1	16	56	200	5.70%	12.81	9*	26	53.6%
2	16	88	316	5.50%	16.00	16*	64	27.3%
3	16	104	391	5.00%	16.00	16*	73	29.8%
4	18	93	361	4.40%	16.53	12*	40	57.0%
5	20	112	116	4.20%	20.00	20*	20	82.1%
6	24	124	428	3.30%	22.57	20*	80	35.5%
7	50	281	732	1.70%	49.50	48*	164	41.6%
8	90	465	3,025	0.85%	81.04	45*	304	34.6%
9	120	620	4,080	0.64%	108.04	60*	409	34.0%
10	240	1,240	8,300	0.32%	216.04	120*	829	33.1%
11	240	1,240	17,573	0.32%	N.A.	94*	1,099	11.4%
12	360	1,860	19,278	0.21%	313.57	150	1,482	20.3%
13	360	1,860	39,101	0.22%	N.A.	75	1,592	14.4%
14	380	1,620	33,664	0.21%	N.A.	87	1,356	16.3%
15	720	3,720	54,053	0.11%	N.A.	280	3,339	10.2%
16	720	3,720	158,156	0.11%	590.60	93	3,307	11.1%

ensures the safety conditions of the runs between the combinations of trains/routes/entrance times. Here, n_S denotes the number of trains, and S_k denotes the set of the allowed combinations of routes/entrance times for a train k. Because of this, the set I is split into disjoint subsets $(S_1, \ldots, S_{n_S})$.

One consequence of formulating the railway feasibility problem as a USPP is: according to first set of constraints, we know that

$$z = \sum_{i \in I} x_i = \sum_{k=1}^{n_S} \sum_{i \in S_k} x_i \leqslant n_S.$$

Thus, we can deduce the parameter $\beta \leqslant n_S$ (see Section 3.2.2).

We experimented with the railway feasibility problem as a USPP, using data from the junction of Pierrefitte Gonesse, north of Paris. In the instances considered (labeled Rail 1 to Rail 16), the time period is generally one hour. The characteristics of these instances are presented in Table 2. This table also shows the value of the linear relaxation (LP), when it is known, as well as the best known solution, with an asterisk indicating the optimal solutions. In those cases where no value is available, due to an internal error reported by Cplex, N.A. is noted in the table. As with the randomly generated instances, the number of non-redundant variables and the reductions obtained are also provided.

5. Computational results

This section presents the computational results obtained for all the instances considered in our study. Both the solutions generated with the Cplex solver and our GRASP implementations are included. All our implementations of GRASP were performed with Ada (Gnat 3.13). The results were obtained on a UltraSPARC-II at 296 MHz for the Cplex, and on a Pentium III at 800 MHz for the GRASP. This difference in equipment prevents comparisons, particularly time comparisons; however, such comparisons are not within the scope of this paper.

We used the MIP-Solver of Cplex 6.0 to solve our instances exactly whenever it was possible. When Cplex could not find the optimal solution within a reasonable time period, we considered the best solution

obtained within that framework. We stopped the Cplex after 50,000 seconds for the random instances and after 200,000 seconds for the railway instances. According to Gondran and Minoux [1], determining clique cuts is generally efficient for solving the SPP problems. In this way, despite the length of time needed to obtain the cliques, the use of clique cuts is more efficient than the use of Cplex with default parameters (particularly given the poor quality of the LP relaxation). In this study, most of the best results for the biggest instances (more than 100 variables) were also obtained when these cuts were set. Moreover, the best bounds were obtained using these cuts. The reduction test algorithm (Algorithm 8) was used prior to the Cplex when the results (or the bounds) obtained could be improved.

We considered the following components in our GRASP implementations:

- The greedy methods described in Sections 3.2.2 and 3.2.3. Preliminary practical tests convinced us that the greedy algorithm described in Section 3.2.1 would require too much computational time to be used effectively here.
- The two choices of α proposed in Section 3.3 with
  - $alphaSet = \{0.0, 0.15, 0.30, 0.45, 0.50, 0.60, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95, 1.0\}$,
  - $|Pool_\alpha| = 5, \forall \alpha \in alphaSet$ (only for reactive GRASP),
  - *probaUpdate* condition: the probabilities are updated every 52 GRASP iterations (only for reactive GRASP),
  - $\delta = 5$ (only for reactive GRASP).
- The local search phase described in Section 3.4.
- The intensification phase described in Section 3.5 with
  - $|Pool| = 5$,
  - A different *Pool* for each α value since some preliminary tests with only one *Pool* for all α values provided inferior results. This concurs with comments made by Glover and Laguna [24] on the use of path relinking as an intensification phase.
- The stopping criterion: 260 GRASP iterations or 18,000 seconds.

Several independent runs of GRASP have been done. Only the best solution for each run is considered. All the results reported for GRASP are the average of these best solutions.

The instances Rail 1 to 7 are small (less than 300 variables and 1,000 constraints), and so Cplex could solve them exactly without any problem. Our experiments shows that the GRASP procedures also provide the optimal solution regardless of the version used. Given that for these instances both Cplex and our experimental method can resolve the problem exactly and without difficulty, we have eliminated further discussion of Rail 1–7 in this paper.

5.1. Resolution with Cplex and bounds

First, we examined both the resolution of our instances with the Cplex solver, and the quality of the lower and upper bounds obtained. These results are presented in Fig. 1. All the values are indicated in comparison with the best known value of each instance (i.e. the best known value is equal to 100% in the figure). The results indicated for Cplex are the best we could obtain. The lower bounds come from a pure greedy algorithm (based on our greedy2 algorithm with $\alpha = 1$), and the upper bounds are the best provided by Cplex. Cplex was not able to provide even one bound for some of the biggest railway instances, due to an internal error. In these cases, we considered β (see Section 4.2) as our upper bound.

We can distinguish three main cases:

- Strongly constrained instances, i.e. with a high density and many constraints as compared to the number of variables (Rnd 1, 2, 9 and 10):

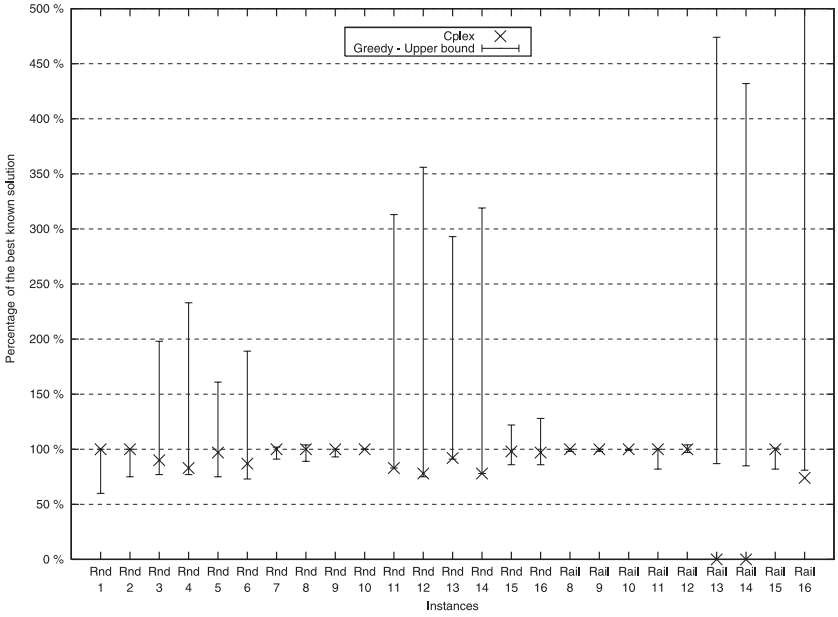

Fig. 1. Resolution with Cplex and bounds.

For strongly constrained instances, Cplex can provide optimal solutions due to the high quality of the upper bound. The greedy procedure is more irregular, sometimes giving results which are sometimes good, sometimes bad, though it does seem to perform better on the larger instances.

- Weakly constrained instances, i.e. with a low density and few constraints as compared to the number of variables (Rnd 7, 8, 15 and 16, Rail 10, 12 and 15):
  For weakly constrained instances, Cplex can provide optimal or at least very good solutions. The greedy results are also good with a good upper bound. However, the gap between the upper and lower bounds seems to increase for the larger instances, which tends to reduce the quality of Cplex results.
- Intermediate instances (Rnd 3–6 and 11–14, Rail 8, 9, 11, 13, 14 and 16):
  For intermediate instances, the Cplex results are far from the best known solutions with the exception of the smaller instances (Rail 8 and 9). Instance Rnd 11 is another exception, only solvable by Cplex using reduction tests. This is natural, given the poor quality of the greedy solutions and the very poor quality of the upper bounds, which are even worse for the larger instances. It would appear that these instances cannot be solved efficiently with the Cplex and thus an exact resolution seems to be compromised.

These results indicate that a heuristic method would be very useful to solve some instances, notably the intermediate instances.

5.2. Impact of each strategy considered for GRASP

In this section, we evaluate the impact of the use of the different strategies proposed. All the results presented in Fig. 2 have been given in comparison with a version of a GRASP based on the greedy2 algorithm where α was randomly selected from the set *alphaSet* for each GRASP iteration. This version, labeled GRASP-Ref, corresponds to a value of 100% in the figure. Three other different versions are presented:

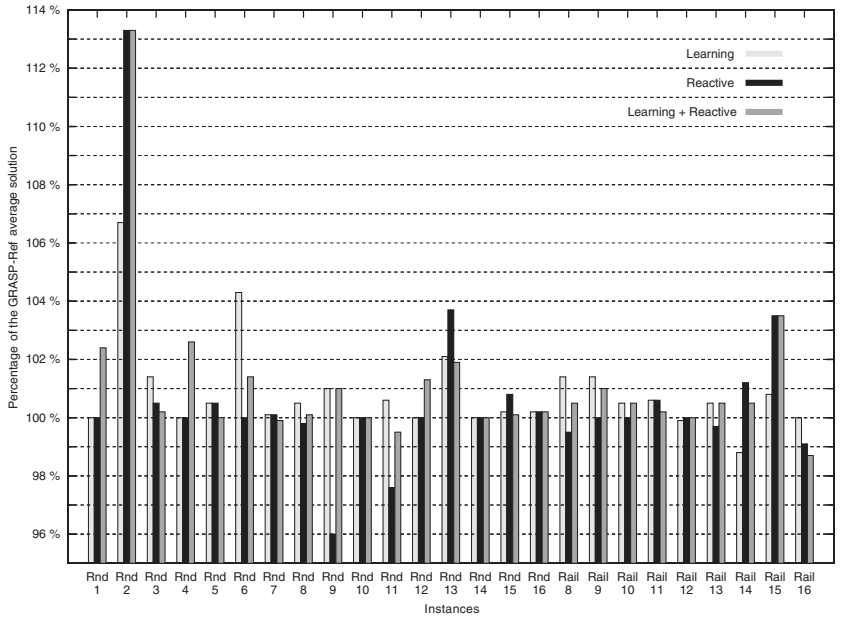

Fig. 2. Impact of each strategy considered for GRASP.

- a version with the greedy3 algorithm, where α was randomly selected from the set *alphaSet* for each GRASP iteration, designed to measure the impact of the learning process alone,
- a version with the greedy2 algorithm and the reactive GRASP, designed to measure the impact of reactive GRASP alone,
- a version with the greedy3 algorithm and the reactive GRASP, designed to measure the impact of both used together.

All the versions presented also contain a local search phase and an intensification phase with path relinking.

The evaluation of the impact of each strategy shows that the learning strategies are efficient for many instances and almost never have a negative impact. On the other hand, reactive GRASP alone provides disappointing results. Its impact is less significant, sinking into the negative for some instances. Except the instance Rnd 2, the average impact of reactive GRASP is nearly null.

The combination of both provides the best average results, even though the combination reflects some of the negative results of reactive GRASP (instance Rnd 11). It is interesting to note that the combination of both strategies can both underperform (instances Rnd 3 and 5) and overperform (instances Rnd 1, 4 and 12) the results of the two strategies taken individually. Actually, it would appear that the learning process can change the significance of some α values during the process, which means that reactive GRASP cannot stabilize the α values' probability. Avoiding this phenomenon (e.g. by waiting for the probabilities to stabilize before using the learning process) would certainly lead to even better results.

5.3. Impact of each phase of GRASP

In this section, the impact of each phase of GRASP (greedy randomized construction, local search, intensification with path relinking) is examined with regard to the best version of our GRASP

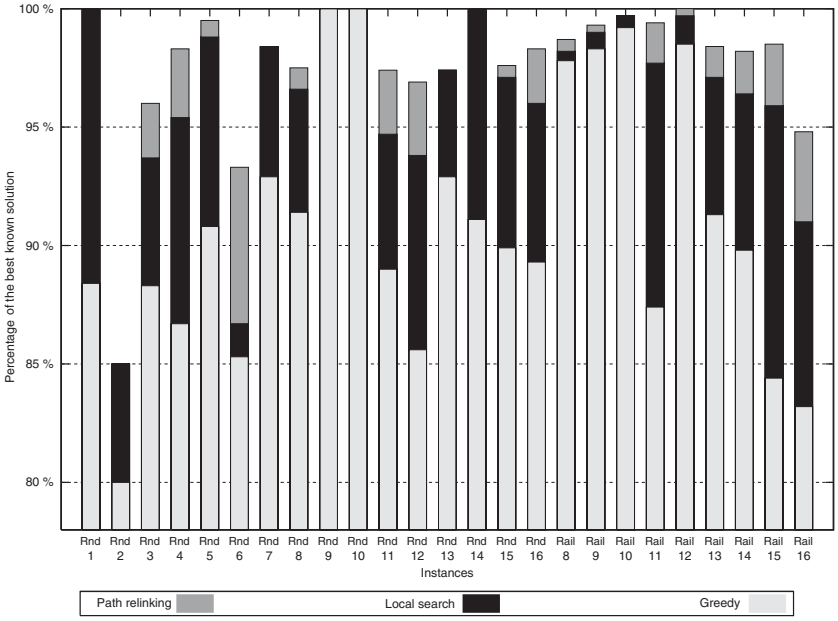

Fig. 3. Impact of each phase of our GRASP procedure.

implementations (i.e. reactive GRASP with the greedy3 and the path relinking algorithms). Fig. 3 presents the average solutions obtained after each phase of our GRASP algorithm in comparison with the best known solution (which is equal to 100% in the figure). Such a comparison permits us to evaluate the importance of each phase in the overall quality of the solutions generated. The time ratio used for each phase is indicated in Fig. 4.

The solutions obtained after the greedy randomized phase can be very good (even optimal for two instances), but are often far from the best known solutions (8.9% on average and up to 20% less than the best known solution). The local search phase improves these results significantly (+5.9% in average). The instances with poor results following the greedy phase often show an improved, though still poor, result following the local search phase (e.g. instance Rnd 6). The path relinking phase permits an additional improvement, over that of the local search phase (+1.5% on average).

These improvements have a cost: together, the local search phase and the path relinking phase use the greater part of the CPU time (respectively 27.5% and 65%). The greedy phase uses more time for the strongly constrained instances (Rnd 1, 2, 9 and 10), due to the time needed to evaluate the variables, and does not ensure better results.

5.4. Solution spectrum of our complete GRASP procedure

Fig. 5 plots the min / max range and the average value observed at the end of each run for all solutions provided, using the best version of our GRASP algorithm. Given that, to our knowledge, there is no other metaheuristic for the SPP available to compare with our GRASP algorithm, we can only compare our results with the best known results.

Our GRASP procedure provides good results for each type of instance (strongly constrained, weakly constrained or intermediate). The average results obtained are only 2.2% less than the optimal solutions (when they are known) and only 2.2% less than the best known solutions. Except for one instance (Rnd 2),

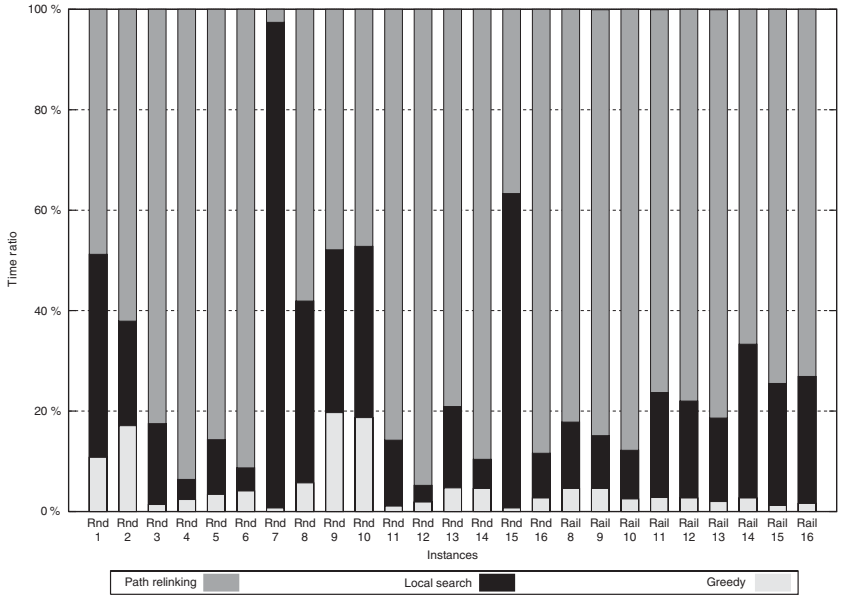

Fig. 4. Time ratio of each phase of our GRASP procedure.

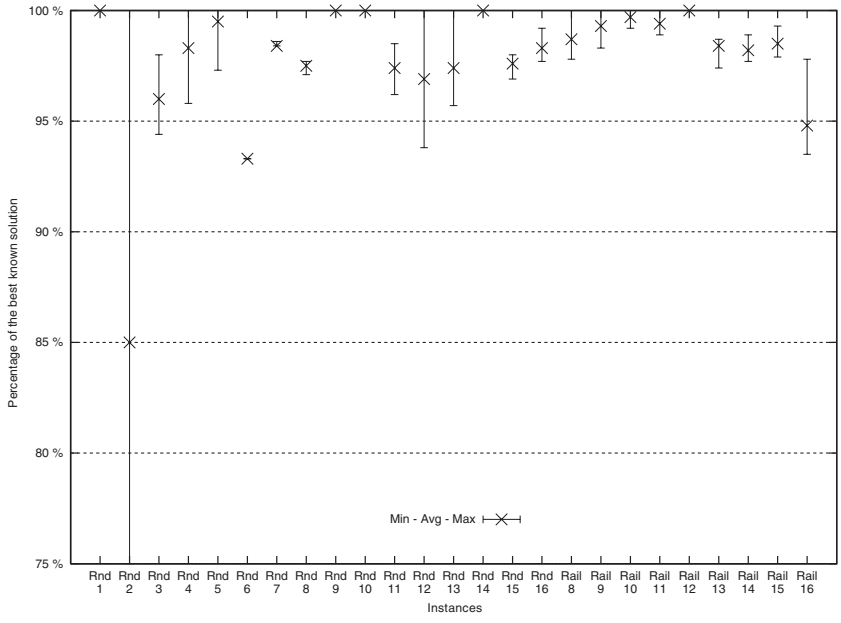

Fig. 5. Computational results of our complete GRASP procedure.

they are, at worst, 6.7% less than the best known solution. For 88% of the considered instances, they are, at worst, 4.0% less than the best known solution. These values indicate a regular solution quality. This is both remarkable and important for the future practical use of the algorithm as solver for railway planning problems.

6. Conclusion and perspectives

In this paper, we have considered a difficult classic combinatorial optimization problem, the SPP, and presented several heuristic algorithms issued from the GRASP metaheuristic and its extensions to solve such problems. The computational results observed show that our GRASP implementation is efficient for solving a large range of SPP. These results seem particularly interesting in terms of the intermediate instances, where exact resolution is not possible within reasonable time constraints, if the well-known commercial software, Cplex, is used.

However, we must admit that even GRASP needs quite a lot of time. In order to improve GRASP performances on SPP problems, we have considered two different possibilities. One possibility would be to reconsider the local search and path relinking algorithms which represent the major part of the computational time. For the local search algorithm, either reducing the research area for 1–2 and for 2–1 exchanges or using a different local search algorithm (tabu search, for example) could improve performances. Stopping the process before the end of the path, as indicated by Resende and Ribeiro [16], might improve the performances of the path relinking algorithm. A second possibility would be to study a node packing problem formulation, though the increase in the size of the problem could cause problems. In addition, modifications could be made to improve our reactive GRASP procedure and to use the learning process and reactive GRASP together more efficiently. Moreover, we could study the impact of some other preprocessing phases to reduce the size of the problem and to improve our upper bounds: reduction tests like those used by Zwaneveld et al. [11] and valid inequalities as defined by Padberg [8] constitute two likely directions for research.

Acknowledgements

The authors wish to thank the anonymous referees whose comments on an earlier version of the paper were very helpful.

References

[1] M. Gondran, M. Minoux, Graphes et algorithmes, Eyrolles, 1995 (in French).
[2] G.L. Nemhauser, L.A. Wolsey, Integer and Combinatorial Optimization, Wiley-Interscience, 1999.
[3] X. Delorme, X. Gandibleux, J. Rodriguez, Application de la métaheuristique GRASP à la résolution d'un problème de capacité d'infrastructure ferroviaire, FRANCORO III, International Conference, May 9–12, 2001, Québec, Canada.
[4] X. Delorme, J. Rodriguez, X. Gandibleux, Heuristics for railway infrastructure saturation, in: ATMOS 2001 (Satellite Workshops of the 28th International Colloquium on Automata, Languages, and Programming, ICALP) Proceedings, Electronic Notes in Theoretical Computer Science, vol. 50, Elsevier Science, 2001, pp. 41–55. Available from <http://www.elsevier.nl/locate/entcs/volume50.html>.
[5] Using the CPLEX callable library (user's guide), version 4.0, CPLEX optimization, 1995.
[6] T.A. Féo, M.G. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Operations Research Letters 8 (1989) 67–71.
[7] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, V.H. Freeman and Company, 1979.
[8] M.W. Padberg, On the facial structure of set packing polyhedra, Mathematical Programming 5 (1973) 199–215.
[9] I. Osman, G. Laporte, Metaheuristics: A bibliography, Annals of Operations Research 63 (1996) 513–623.
[10] M. Rönnqvist, A method for the cutting stock problem with different qualities, European Journal of Operational Research 83 (1995) 57–68.
[11] P.J. Zwaneveld, L.G. Kroon, H.E. Romeijn, M. Salomon, S. Dauzère-Pérès, S.P. Van Hoesel, H.W. Ambergen, Routing trains through railway stations: Model formulation and algorithms, Transportation Science 30 (3) (1996) 181–194.
[12] S.-H. Kim, K.-K. Lee, An optimization-based decision support system for ship scheduling, Computers and Industrial Engineering 33 (1997) 689–692.

[13] A. Mingozzi, V. Maniezzo, S. Ricciardelli, L. Bianco, An exact algorithm for the project scheduling with resource constraints based on a new mathematical formulation, Management Science 44 (5) (1998) 714–729.
[14] F. Rossi, S. Smriglio, A set packing model for the ground holding problem in congested networks, European Journal of Operational Research 131 (2001) 400–416.
[15] L.S. Pitsoulis, M.G. Resende, Greedy randomized adaptive search procedures, in: P. Pardalos, M. Resende (Eds.), Handbook of Applied Optimization, Oxford University Press, 2002, pp. 168–183.
[16] M.G. Resende, C.C. Ribeiro, Greedy randomized adaptive search procedures, in: F. Glover, G. Kochenberger (Eds.), Handbook in Metaheuristics, Kluwer Academic Publishers, 2002, pp. 219–249.
[17] P. Festa, M.G. Resende, GRASP: An annotated bibliography, in: C.C. Ribeiro, P. Hansen (Eds.), Essays and Surveys on Metaheuristics, Kluwer Academic Publishers, 2001, pp. 325–367.
[18] X. Delorme, Optimisation combinatoire et problèmes de capacité d'infrastructure ferroviaire, Mémoire de DEA, Université de Valenciennes et du Hainaut Cambrésis, Valenciennes, France, 2000 (in French).
[19] T.A. Féo, M.G. Resende, Greedy randomized adaptative search procedures, Journal of Global Optimization 6 (1995) 109–133.
[20] X. Gandibleux, D. Vancoppenolle, D. Tuyttens, A first making use of GRASP for solving MOCO problems, 14th International Conference in Multiple Criteria Decision-Making, June 8–12, 1998, Charlottesville, USA.
[21] D. Vancoppenolle, Résolution par GRASP de problèmes d'optimisation combinatoire, Mémoire de 2ème licence en informatique, Université de Mons-Hainaut, Mons, Belgique, 1998 (in French).
[22] T.A. Féo, M.G. Resende, S.H. Smith, A greedy randomized adaptative search procedure for maximum independent set, Operations Research 42 (1994) 860–878.
[23] M. Prais, C.C. Ribeiro, Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment, INFORMS Journal on Computing 12 (2000) 164–176.
[24] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, 1997.
[25] M. Laguna, R. Martí, Grasp and path relinking for 2-layer straight line crossing minimization, INFORMS Journal on Computing 11 (1999) 44–52.
[26] M.G. Resende, C.C. Ribeiro, A GRASP with path relinking for permanent virtual circuit routing, Tech. rep., AT&T Labs Research, 2001.