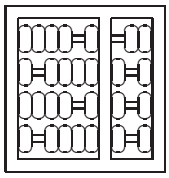


INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Constraint Programming and GRASP Approaches to Schedule Oil Well Drillings

Romulo A. Pereira *Arnaldo V. Moura*
Cid C. de Souza

Technical Report - IC-05-001 - Relatório Técnico

January - 2005 - Janeiro

Constraint Programmin and GRASP Approaches to Schedule Oil Well Drillings

Romulo Albuquerque Pereira * Arnaldo Vieira Moura†

Cid Carvalho de Souza‡

Abstract

In the process of drilling oil wells, it is necessary to schedule resources, such as oil derricks and boats, in order to execute development activities that prepare promissing wells for oil extraction. The scheduling of such activities must satisfy a variety of constraints and must attain some objective criteria, such as maximizing oil production. In this paper, we discuss a greedy randomized adaptive search procedure (GRASP) algorithm for the scheduling of oil well drilling activities. We describe in detail our GRASP implementation and compare the results obtained with results derived from a well accepted constraint programming implementation. Computational experience on real instances of the problem indicates that the GRASP implementation is competitive, outperforming the contraint programming implementation.

*Institute of Computing, University of Campinas, 13081-970 Campinas, SP

†Institute of Computing, University of Campinas, 13081-970 Campinas, SP

‡Institute of Computing, University of Campinas, 13081-970 Campinas, SP

Contents

List of Figures

1 Introduction

Oil extracted from oceanic basins is an increasingly important fraction of the total world offer of petroleum. The Brazilian Petrobras is one of the leading companies on deep seas oil exploration, and one of the twenty biggest oil companies in the world. It explores diverse petroliferous basins, each with hundreds of promising spots where productive oil wells could be located. However, these places need to be developed before they are turned into productive wells. In order to accomplish the activities involved in oil well development, Petrobras uses oil derricks and ships, some rented, others of its own property. These resources are limited and expensive, either in acquisition value or rent value, and must, therefore, be used efficiently.

The oil *well drilling problem* (WDP) can be summarized thus: given a set of wells, given the activities to be executed in each well and the available resources, determine a scheduling of the activities to be carried by each resource, within a given time horizon, in such a way as to optimize some objective criteria. This scheduling must also take into account several engineering and operational constraints. In this work, the WDP is studied, taking into account the specificities faced by Petrobras when exploring oil in deep sea basins. The engineering and operational constraints will be presented in detail, and an heuristic strategy will be developed in order to maximize oil production within a given time horizon.

Similar problems have been studied in the literature [9]. However, the WDP at Petrobras imposes more constraints than those already considered in other works. In fact, some of these new constraints have been listed as topics for future research [9]. In this article we deal with these new, more realistic, constraints.

The remainder of this paper is organized as follows. The next section describes the WDP in more detail. Section 3 presents the constraint programming model developed by Petrobras to solve the problem. Section 4 discusses our GRASP approach to the WDP and describes our new algorithm. Section 5 presents the results obtained with the new GRASP implementation. Finally, conclusions are drawn in the last section.

2 The Well Drilling Problem

When a location is considered a promising oil well, boats and oil derricks are sent there to carry through the due operations of *drilling*. The development of a well, from determining its location until it is put into production, involves several stages.

After the well is drilled, the preparation for petroleum extraction is initiated (a stage called *completion*). First, the "Wet Christmas Tree"[1] (WCT) is placed at the mouth of the well so that oil does not leak to the sea. In this stage, oil derricks are used. Later, a boat takes a pipeline, of which an extremity is connected to the WCT, and connects the other extremity to a manifold[2], or else connects it up directly to the surface into a production platform (this stage is called *interconnection*). A manifold is installed by a boat and its use prevents the need for exclusive pipelines connecting each well to the surface. In the

[1] A complex metallic structure where hydraulic valves are attached.

[2] A manifold is a structure for the junction of pipelines in the deep of the sea.

absence of manifolds, linking each well directly to the surface would render the process as excessively costly, since the cost of such pipelines, per unit length, is very high. Once the interconnection stage is completed, the extraction of petroleum can be started. For that, Stationary Units of Production (SUPs) are anchored at specific locations in the surface, and a boat interconnects the manifold to a SUP. A SUP processes and stores, when it has storage capacity, the extracted products. Later, ships come to collect the products, fetching them to land, storage sites or other processing units. If the oil outflow is very high and the SUP does not have storage capacity, then a petroliferous platform may be installed at the surface.

The main constraints involved in the process of scheduling the necessary activities for the development of a oil well are:

1. *Technological Precedence*: sets an order among the activities. There are four categories of technological precedence:
   (a) FS(A,B): Activity A must finish after the start of activity B.
   (b) SS(A,B): Activity A must start after the start of activity B.
   (c) SF(A,B): Activity A must start after the finish of activity B.
   (d) FF(A,B): Activity A must finish after the finish of activity B.
2. *Mark-Activity*: an activity must finish before or initiate after a determined date, or *mark*, with or without lag time. This date is generally related to some external event, *e.g.*, the installation of a petroliferous plataform.
3. *Baseline*: sets the start and end dates of an activity.
4. *Constraints in the Use of Resources*: to execute an activity, due to its intrinsic nature, one must use a resource that matches some operational characteristics. For an activity that requires a boat, one must verify if the on-board equipments can operate at the specified depth. For an activity that requires an oil derrick, one must verify:
   (a) Its type: anchored, SSDP, NSDP or oil derrick-ship.
   (b) Its abilities: TOP DRIVE, HPHT, BOP 16, BOP 18, Drilling.
   (c) Its maximum and minimum depth of operation.
   (d) Its maximum depth of drilling.
5. *Concurrence*: two activities in the same well can not be executed simultaneously. Similarly, two activities can not be executed by the same resource simultaneously.
6. *Unavailability*: resources may be unavailable for a period of time, either for maintenance reasons or due to contract expiration.
7. *User Defined Well Sequences*: the user can specify a desired sequence for the activities of drilling or for the activity of "production beginning" of different wells. The sequence is an ordered list of one or more wells such that if well A precedes well B in the list,

then the activity F_A of well A must finish before the start of the activity S_B of well B. The activities F_A and S_B are either activities of "drilling" or activities of "production beginning" of their respective wells, according to the type of the sequence. These sequences are specified by engineers in order to avoid loss of pressure in an oil field.

8. *Surface Constraints*: also known as *killer bubbles*, these constraints represent a security area, defined by the user around each well, so that oil derricks used in the activities in those wells do not collide. The restricted area is specified by a closed polygon defined by coordinates around the well. When the center of the first well is inside the restricted area of the second well, the activities executed by oil derricks in these wells cannot be simultaneous. These constraints must be verified between each pair of mobile oil derricks, and each pair of mobile and anchored oil derricks.

9. *Cluster Constraints*: an activity can be part of a cluster, which is a set of activities that must use the same resource.

10. *Same Oil Derrick for a Well*: to avoid unnecessary displacements of oil derricks between wells, it is desirable that the same oil derrick executes as much of the activities of a well as possible.

The oil production of a well is calculated as follows. Each well has an associated outflow and an activity whose purpose is to mark the beginning of its production. When this last activity is concluded, the well is considered in production. The production is measured by multiplying its oil outflow by the time remaining since its beginning of production until the established horizon. If the beginning of production activity is executed after the time horizon, the corresponding well production is not considered.

The objective is to obtain a scheduling for all the activities, satisfying all the constraints and maximizing oil production. Other objectives to be attained are:

1. Generate faster solutions. Human made solutions take many hours, even days to be constructed. A faster method would permit the analysis of different scenarios for the same problem, for example, by adding or removing resources. Furthermore, alterations in plans would not result in new hours or days spent in rescheduling.

2. With scheduling automation, highly skilled engineers once responsible for the manual scheduling can be reallocated to other areas in the company.

By the problem description, together with its constraints, it can be observed that, besides dealing with a real application, it also involves several particularities that makes it sufficiently differentiated from similar problems reported in the literature.

3 A Constraint Programming Solution

In 2000, a project team of Petrobras decided to model the WDP using the ILOG Constraint Programming libraries ILOG Solver and ILOG Scheduler. These tools allow for easy model-

ing and, being C++ native, their use is simple. After four years of development, alterations, corrections and improvements, the developed tool, named ORCA[3], pleased its idealizers.

The next subsection describes the constraint programming paradigm used in ORCA. The model and its associated algorithms are shown in the subsequent subsections.

3.1 Constraint Programming

In 1987, the work of Jaffar and Lassez [29] established the theoretical basis by which diverse constraint programming languages could be developed. The basic idea is to substitute the mechanism of traditional logical inference (unification) for a more generic and efficient one, known as *constraints handling*. This mechanism has been used since the 80's in solving *Constraint Satisfaction Problems* (CSPs) in Artificial Intelligence and other fields [33, 32, 27, 52, 53]. Briefly, a CSP consists of:

- A set of *variables* $X = \{x_1, ..., x_n\}$;
- For each variable x_i in X, a set of values, D_i, called its *domain*;
- A set of *constraints*, C, restricting the values that each variable in X can take.

A solution to a CSP is an assignment of values to the variables in X, in such a way that all constraints in C are satisfied. Many problems in Operational Research, as diverse as planning, allocating resources, managing time, organizing personnel, cutting materials, blending mixtures, assigning radio frequencies, among other combinatorial problems, can be represented as CSPs. A *constraint program* (CP) is the embodiment of a CSP in a particular *constraint programming language* (CPL).

A CP generally follows the structure presented in Figure 1. Notice that this structure reflects the corresponding CSP items.

<Declaration of Variables and Domains>
<Declaration of Constraints>
<Search for Solutions (*labeling*)>

Figure 1: Basic structure of a program in Constraint Programming.

The variables with its domains define the solution space. The constraints establish *relations* between the variables, limiting the values they can assume concomitantly, thereby reducing the search space. At each iteration, the domains are reduced by force of the constraints. This reduction is obtained by a mechanism called *constraint propagation*, whose function is to guarantee the *partial consistency* of the system as a whole. The constraint propagation mechanism works in the following way. When the domain of a variable is modified, *e.g.* one of its elements is removed, this information is transmitted, or propagated, to all the variables associated to it by one or more constraints. For example: let X and Y

[3]Portuguese acronym for "Optimization of Critical Resources in the Production Activity".

be integer variables with the domains $D_X = D_Y = [1, 10]$. Imposing the constraint $X < Y$, the domains are changed to $D_X = [1, 9]$ and $D_Y = [2, 10]$. Suppose now that the constraint $X \geq 5$ is imposed. This makes $D_X = [5, 9]$, and the domain of Y is automatically changed to $D_Y = [6, 10]$.

In general, only partial consistency is guaranteed, since there is no known polynomial time algorithm that enforces *total consistency* in a generic CSP[4]. Moreover, for efficiency reasons, some algorithms used to detect constraint violations are not able to identify certain types of inconsistent states. One of these algorithms, called *arc-consistency*, works as follows: let A and B be two variables restricted by the constraint r, and with domains, respectively, D_A and D_B. A value $v_A \in D_A$ has *support* in the domain D_B if, when attributing v_a to A, there is a value $v_B \in D_B$ that can be set to B without violating r. The arc-consistency algorithm removes from the domain of each variable all those values that do not have support in the domain of any other variable associated with it by some constraint [36].

Next, the *labeling* process starts as indicated in Figure 1. It begins by selecting a variable and assigning to it one of its domains values. The order of choice of the variables and of the domain values is flexible and affects the performance of the algorithm. After setting the value to the chosen variable, the propagation mechanism reduces the other domains. In case some inconsistency is detected, the last value attribution has to be undone, a mechanism known as *backtracking*, and another alternative is taken. This process repeats itself until a solution is found, or until the nonexistence of a viable solution is proved [36, 33].

In some situations, it is not enough to search for a viable solution, but it is also necessary to find one that minimizes (or maximizes) some objective criteria. The concept of optimization can be embodied in the constraint programming paradigm in the following way. When a solution is found, its cost c is calculated and a new constraint is inserted in the CP indicating that the cost of the next solution will have to be less than c (assuming minimization). This process is repeated until no more viable solutions can be found, or until a time limit is reached. If no more viable solutions can be found, the last solution is one of optimum cost [36, 33].

A great challenge in constraint programming is the creation of a model that adequately represents the real conditions of a problem. A poor model, for example, can generate erroneous or not satisfactory solutions, or can even lead to no solution at all.

3.2 The ORCA Implementation

The WDP, described in Section 2, was modelled using ILOG Solver and ILOG Scheduler libraries. In this approach the problem representation is dissociated from the algorithms used to solve it, thus offering easiness of development, of understanding and of adaptation.

The model consists of two types of variables, both with integer domains:

- One type represents the beginning of execution of each activity in its well. It is characterized by a minimum start time and a maximum start time, both of which must have the same value so that the activity may be considered scheduled.

[4]CSPs belong to the class of NP-complete problems [36].

- Another type that represents which resource will execute each activity in its well. It is characterized by a set of the possible resources, one of which must be chosen to execute the activity.

All the constraints described in Section 2 are imposed on the described variables using built-in methods of the ILOG classes. For example, let a and b be activities represented by objects of the ILOG class *IlcActivity* and let *P1* be the model represented by the ILOG class *IlcManager*, we constrain a to start after the finish of b through the following code:

P1.add(a.startsAfterEnd(b));

The mechanism of constraint propagation is handled directly by the ILOG libraries. However, in many cases, simple constraint propagation is not sufficient to get a unique value for each one of the variables. In other words, constraint propagation and domain reduction may not be enough to find a unique solution to the problem. Thus, it is necessary to search for solutions.

The ILOG libraries provide not only a generic search algorithm, but also ways to develop the user's own search method. With the ILOG Solver, the implementation of search algorithms is based on the idea of *goals*. Goals make it possible to implement algorithms where the exact sequence of operations is not known in advance. In other words, one can use goals to guide a search even though it is not know ahead of time exactly how the search will be executed. This is often called *non-deterministic programming* in the sense that its precise steps are not determined nor predicted in advance, yet the programming does have a definite purpose or goal in view.

The search mechanism exploits the following algorithm: as long as there exist variables not yet assigned,

1. choose one of those variables;
2. choose a value to assign to this variable;
3. propagate the effects of that assignment.

In general, it is not known which value in the domain of the variable is consistent with all the constraints. If that binding leads to inconsistencies, it must be undone, and another value must be tried. The ILOG Solver also offers predefined ways to find solutions that minimize or maximize some objective criteria. For that, it uses a *Branch and Bound* [8, 35] technique.

The ILOG Solver implements an implicit representation of the search tree, consisting of a sequence of left and right decisions. The search moves from one position in the tree to another through a combination of backtracking and recomputation of decisions [6]. The Solver's search engine generally works like this: open nodes, that is, those nodes that can still be explored, are stored in a heap. At each step, the Solver chooses the best open node with respect to a given evaluation function and moves to that node in order to expand it. It implements predefined evaluation functions for the following search techniques:

- *Depth First Search (DFS)* is the standard search procedure of the ILOG Solver. DFS considers out-going edges of a vertex before neighbors of that vertex. The real problem with DFS is that it cannot recover from early poor choices. A bad decision early in the search may lead the search down through a large number of non-goal states.
- *Best First Search (BFS)* is a search algorithm which optimises *Breadth First Search*[5] by ordering all current paths according to some heuristic. The heuristic attempts to predict how close the end of a path is to a solution. Paths which are judged to be closer to a solution are extended first. It is fully described in [38]. The Solver implementation uses a parameter ϵ. When selecting an open node, the Solver determines the set of open nodes whose cost is at most $(1+\epsilon)$ worse than the best open node. If the child of the current node is in this set, the Solver goes to this child. If not, the Solver chooses the best open node.
- *Limited Discrepancy Search (LDS)* was first defined in [26]. The discrepancy of a search node is defined as its right depth, that is, the number of times the search engine has chosen the right branch of a choice point when going from the root of the search tree to the current node. Given a parameter k, the discrepancy search procedure is one that will first explore nodes with a discrepancy less than k. After this exploration is complete, it will explore nodes with a discrepancy between k and $2k$, and so on. In fact, this search procedure cuts the search tree in strips.
- *Depth Bounded Discrepancy Search (DDS)* was introduced in [48]. A variation of LDS, DDS makes the assumption that mistakes are more likely made near the top of the search tree than further down. For this reason, this procedure does not count the number of discrepancies but the depth of the last one. The ILOG Solver implements an improved version of Walsh's schema [48]; it does not count the depth of the last discrepancy but of the *d-th* last, where d is a parameter of the search.
- *Interleaved Depth First Search (IDFS)* was introduced in [37]. IDFS tries to mimic the behavior of an infinite number of threads exploring the search tree. The Solver implements a variation that limits the depth of this interleaving behavior.

All these search procedures were tested over real instances of the WDP. Pure BFS proved to be inefficient for the problem. In almost all scenarios it did not find any solution within the first hour of execution. The DFS and IDFS schemes showed some progress, but the best results were found using LDS or DDS. In the tests, LDS surpassed DDS slightly, besides being more flexible. That is because DDS is more efficient if the search heuristic is very good, *i.e.*, if it makes mistakes only in the top of the search tree [28], which is something hard to achieve. This tests were run under the plataform described in Section 5.

Let us now present the goals created to guide the scheduling process. They are executed in the order they are presented:

[5] Breadth first search is a tree search algorithm that starts at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, explore their unexplored neighbor nodes, and so on until it finds the goal.

1. **Choosing the resources:** This goal is responsible for choosing the resources that will execute the activities. It simply selects the available resource less used until then for each activity.

2. **Handling sequences of drilling:** In case there are sequences of drilling, this goal sets the start execution date of the drilling activities in each sequence. When ordering the activities, it selects the available variable that is not yet set among all the variables present in the sequences of drilling and that generates the greater increase in oil production.

3. **Scheduling production activities:** This goal sets the start execution date of the beginning of production activities. When ordering the activities, it selects the variable not yet set as follows:

   - Firstly, the one available among all variables present in the sequences of production, and that generates the greater increase in oil production.
   - Secondly, that available variable which generates the greater increase in the oil production.

4. **Scheduling the other activities:** This goal sets the start execution date of all the other activities. When ordering the activities, it selects the first one not yet set.

Nowadays the ORCA solver is often used by engineers, both to define a good schedule for the activities of some oil fields. It is also used to analyze the need for acquiring or renting new resources. Engineers confirmed that ORCA generates better solutions than those made by humans. In one of its uses in a real instance of the problem, ORCA showed that buying a third oil derrick was unnecessary and that it was better to add a new LSV ship instead. As a result, ORCA assured a net gain of US$15 million to Petrobras, and anticipated oil production by 26 days.

4 A GRASP Implementation

As previously seen, the ORCA solver produced good results. In order to test if other approaches could lead to better solutions, and also to compare results, other applicable techniques were considered. Amongst them, *Tabu Search* [24, 25, 20, 21, 23, 30, 22], *Ant Colony* [11, 10, 5] and *GRASP* [40, 46, 4, 12, 47, 41, 18]. *Tabu Search* is a well studied metaheuristic, and it was used in a simpler version of this same problem [9]. However, some issues proved to be particularly difficult to treat, especially the definition of an adequate neighborhood and ways to explore it. *Ant Colony* has been successfully applied to *job-shop scheduling* problems [7], but with no exceptional results [10]. GRASP was the approach that seemed most appropriate for the WDP, it being a well studied metaheuristic that has been successfully applied to scheduling problems [1, 13, 16, 31, 19, 14, 2, 17, 49, 34, 43, 44, 50, 42, 3]. Furthermore, 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 few parameters.

The next subsection describes GRASP, the paradigm used to construct a new implementation, named GRASPS. The model and its algorithms are shown in the subsequent subsections.

4.1 Greedy Randomized Adaptive Search Procedure (GRASP)

In this paper, we apply GRASP ideas to the problem of scheduling the activities of oil wells. The GRASP methodology is an iterative process, where each iteration consists of two phases: **Construction** and **Local Search** [40, 46, 4, 12, 47, 41, 18, 15]. The **Construction** phase builds a greedy randomized feasible solution, whose neighborhood is explored afterward in the **Local Search** phase. This process is repeated many times and the best overall solution is returned as the result.

```
1: procedure GRASP(ListSize, MaxIter, Seed)
2: for k = 1 a MaxIter do
3:    Solution ← Construct_Solution(ListSize, Seed);
4:    Solution ← Local_Search(Solution);
5:    Update_Solution(Solution, Best_Solution_Found);
6: end for
7: return Best_Solution_Found;
8: end GRASP
```

Figure 2: Pseudo-code of the GRASP Metaheuristic.

Figure 2 illustrates a generic implementation of the main block of GRASP, in pseudo-code. The input for GRASP includes parameters for setting the candidate list size (*ListSize*), the maximum number of GRASP iterations (*MaxIter*), and the seed (*Seed*) for the random number generator. The GRASP iterations are carried out in lines 2-6. Each iteration, as already mentioned, consists of the construction phase (line 3), the local search phase (line 4) and, if necessary, the incumbent solution update (line 5). In the construction phase, a feasible solution is built, updating the variable *Solution*. Then the local search algorithm seeks a better solution in the neighborhood of *Solution*, aiming at the minimization or maximization of a given criterion, and updates *Solution*. If *Solution* is better than the one stored in *Best_Solution_Found*, the function *Update_Solution* replaces the latter. This process of construction, search and update is executed *MaxIter* times.

In the construction phase, a feasible solution is built, one element at a time. Figure 3 illustrates a generic implementation of the construction phase of GRASP, in pseudo-code. Input for the algorithm includes the candidate list size (*ListSize*) and the seed (*Seed*) for the random number generator. The construction phase iterations are carried out in lines 2-8. At each construction iteration, the next element to be added is determined by ordering all elements in a candidate list with respect to a greedy function that measures the, maybe myopic, benefit of selecting each element. This list is called the Restricted Candidate List

(*RCL*). The adaptive component of the heuristic arises from the fact that 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 elements. The probabilistic component is characterized by randomly choosing one of the best candidates in the RCL, but usually not the best one. This way of making the choice allows for different solutions to be obtained at each iteration, while not necessarily jeopardizing the adaptive greedy component. The solutions generated by a GRASP construction phase are not guaranteed to be locally optimal. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution.

```
1: procedure Construct_Solution(ListSize, Seed)
2: Solution ← 0;
3: Evaluate the incremental costs of the candidate elements;
4: while Solution is not a complete solution do
5:    Build the restricted candidate list, RCL(ListSize);
6:    Select an element s from the RCL at random;
7:    Solution ← Solution ∪ {s};
8:    Reevaluate the incremental costs;
9: end while
10: return Solution;
11: end Construct_Solution
```

Figure 3: Pseudo-code of the Construction Phase of GRASP.

In the local search phase, a deterministic local search algorithm works in an iterative fashion, by successively replacing the current solution by a better solution from its neighborhood. This algorithm seeks to optimize the solution built in the construction phase through a local search for a better neighbor s', that is, one with a lower (minimization) or higher (maximization) value according to a objective function. When s' is found, *Solution* is replaced by it. This process is iterated until when there is no better solution in the current neighborhood with respect to some objective function. Figure 4 illustrates a generic implementation of the local search phase of GRASP, in pseudo-code. Input for the algorithm includes the solution built in the construction phase (*Solution*). The local search phase iterations are carried out in lines 2-4.

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

Figure 4: Pseudo-code of the Local Search Phase of GRASP.

The GRASP metaheuristic has been successfully applied to a wide variety of optimization problems, including routing, covering and partition, location, minimum Steiner tree, optimization in graphs, assignment, timetabling, scheduling, manufacturing, transportation, power systems, telecommunications, graph and map drawing, and VLSI, among other areas of application [40]. For a detailed bibliography, see [18, 39].

The next subsection shows details of the GRASP implementation, called GRASPS, developed in this work.

4.2 The GRASPS Implementation of GRASP

The GRASPS implementation was constructed using a C/C++ programming language and by defining appropriate classes for the problem. As in the ORCA model, there are two types of integer variables:

- One type represents the beginning of execution of each activity in its well. It is characterized by a minimum start time and a maximum start time, both of which must have the same value so that the activity may be considered scheduled.
- Another type that represents which resource will execute each activity in its well. It is characterized by a set of the possible resources, one of which must be chosen to execute the activity.

All the constraints described in Section 2 were imposed on the variables. Three constraints, namely, *Mark-Activity Precedence*, *Baseline* and *Constraints in the Use of Resources*, were set while reading the problem data, before the search begins. Note that, in these cases, all values needed to set the constraints are already defined. The other seven constraints were dealt with during the search for solutions, the variables involved being assigned single values.

The main procedure of GRASP, illustrated in Figure 2, was kept almost intact. The following simple adaptations were made:

- The search procedure was interrupted by a time limit instead of by the number of iterations.
- The parameter *ListSize* is updated from time to time during the search. The value of *ListSize* is decremented or incremented by some value whenever a predefined amount of time is reached with no improvement of the best solution. This, respectively, allows us to focus into a greedier heuristic or to explore larger regions of the search space. With this scheme we obtain a *dynamic sized RCL* in opposition to the original *static sized RCL*.

The objective criteria was the same one used by the ORCA implementation (see Section 4). In other words, we seek solutions with the highest oil production.

In the construction phase, illustrated in Figure 3, a number of modifications were introduced. They are listed below.

1. The first time ever the construction phase is initiated, we use a *ListSize* equal to one, which makes the algorithm behave like a pure greedy heuristic. Note that if there are few constraints obstructing the greedy heuristic, it tends to generate a good or even very good solution. For example, in two of twelve instances tested, the best solution was found in the first solution of the construction phase.

2. Before line 3 of the code shown in Figure 3, we added a function called *isPressed*. This function verifies if any activity of any well has a start time of small domain and if only one resource can do that activity at that time. By a start time of small domain we mean that its minimum and maximum start times are very close. So having only one resource able to execute it at that period, there is less flexibility to schedule that activity. If there are such activities, the function schedules their wells. Note that retarding the schedule of such activities could render the solution infeasible, as other activities may occupy the period of time where those activities would be scheduled. We schedule the wells and not only the activities in order to comply with the *Same Oil Derrick for a Well* constraint (see Section 2).

3. The candidates are defined by the production wells that are available (meaning that there are no wells yet not scheduled which must precede them), or the injection wells that have activities of production wells succeeding them. The activities of injection wells that do not have activities of production wells succeeding them are left to be scheduled after all others. Note that injection wells do not contribute to the oil production and therefore must not be scheduled before production wells, unless there are constraints forcing such a schedule. We schedule the wells and not only an activity because of the same *Same Oil Derrick for a Well* constraint (see Section 2).

4. The evaluation of incremental costs (line 3 of Figure 3) accesses how much oil a well can produce until the end of the time horizon. The RCL is built with those wells that can offer the highest yields of oil. Actually, not only the oil offer is considered, but also the oil offer of the constrained successors of that well.

5. 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 (line 5 of Figure 3). The elements of the RCL are assigned equal probabilities of being chosen. However, any probability distribution can be used to bias the selection towards some particular candidates [40]. We tried to bias the selection of the candidates proportionally to their oil offer.

6. To schedule the candidate well (line 7 of Figure 3), the following algorithm was used:

   As long as there exist activities not yet scheduled in the well:

   (a) choose any activity available in the well, *i.e.*, an activity not yet scheduled and such that there is no other activity not yet scheduled in the wells that precede it.

(b) choose a resource to execute this activity that can actually execute it and that can finish it in the earliest possible time.

(c) set the start time of the activity as the earliest possible time, *i.e.*, the maximum between the earliest time the resource is available to execute the activity and the minimum start time of the activity.

(d) all activities that are constrained to succeed it have their minimum start times updated according to constraints associated to it.

The scheduling of the well is done trying to satisfy all constraints, especially those seven constraints not yet in operation. If any constraint is violated (a test is performed after each activity is scheduled), the construction of this solution is aborted and the construction of a new one is started. Of course, we could backtrack one or more steps instead of starting again, but this could make this phase take too much time, especially if the first steps were wrong.

7. After a well is scheduled, all activities that are constrained to succeed it have their minimum start times updated according to constraints associated to the scheduled well. If any constraint is not satisfied or the minimum date an activity can be scheduled is higher than its maximum start time, the construction of this solution is aborted and the construction of a new one is started.

Since there is no guarantee that the schedule obtained in the construction phase is locally optimal, a local search procedure is used to improve the solution.

The main block of the *local search phase*, illustrated in Figure 4, was kept just the same. A local search algorithm was chosen and an appropriate neighborhood was defined, so as to permit fast explorations that could quickly lead to better solutions. The local search algorithm employed in this GRASP is the two exchange local search based on the disjunctive graph model of Roy and Sussmann [45]. This same algorithm was used in [3], but for a Job Shop Scheduling problem (JSP).

It is easy to see the relation between the problem considered here and the JSP. The JSP can be defined as follows: a set M of machines, a set J of jobs and to each job $j \in J$ it is associated an ordered collection of operations $t_k[j]$, $1 \leq k \leq n_j$, where n_j is the number of operations associated with job j. To each operation t (t is short for $t_k[j]$) is associated a duration $l(t)$ and a machine $m(t)$. The problem is to find a scheduling for the operations that optimizes a given objective criteria. This scheduling must obey the order of operations in the jobs and must also not allow that the same machine execute more than one operation simultaneously. The relation between the JSP and the WDP can be seen by considering the jobs as the oil wells, and the operations of the jobs as the activities of the wells. The precedence relationship among the activities of a well corresponds to the order of operations in the jobs. The WDP resources are the machines who execute the operations in the JSP. Note, however, that the WDP is a more complex problem, in the sense that the operations may not have a specific machine to process it. Furthermore, a WDP solution must satisfy many constraints and the objective criteria is to maximize oil production.

In order to apply the two exchange local search to the WDP, we swap two elements in the schedule. For example, if we could swap elements A and B and the schedule in one resource was like $... \rightarrow A \rightarrow ... \rightarrow B \rightarrow ...$, where $\rightarrow$ represents a conjunctive arc, the result of the swap would be a schedule like $... \rightarrow B \rightarrow ... \rightarrow A \rightarrow ...$ in that same resource. Note that the exchanged elements need not be in the same resource. If A and B were in different resources, for example in resource X the schedule was $... \rightarrow X1 \rightarrow A \rightarrow X2 \rightarrow ...$ and in resource Y was $... \rightarrow Y1 \rightarrow B \rightarrow Y2 \rightarrow ...$, the result of the swap would be a schedule like $... \rightarrow X1 \rightarrow B \rightarrow X2 \rightarrow ...$ and $... \rightarrow Y1 \rightarrow A \rightarrow Y2 \rightarrow ...$, respectively, in resources X and Y. Note also that the execution time of elements A and B may be different, and so all activities after them may have to have their start times updated.

We need to decide, of course, what an element stands for. Some options are:

- Activity: Used in [9] in a simpler version of this problem. Very small granularity, which would lead to huge neighborhoods. Moreover, most of the time, moving an activity to another position would force us to move also its predecessors and successors in the same well, because of the *Same Oil Derrick for a Well* constraint (see Section 2).

- Well: Higher granularity. The problem is that the sequence of activities in wells may be splited in the current schedule (*e.g.*: $... \rightarrow A \rightarrow ... \rightarrow A \rightarrow ...$) because of *Technological Precedence* constraints (see Section 2), and so moving all parts would take time in verifying many constraints. Worse, exchanging the whole well may not be possible, even though exchanging only some activities of it could be.

- Part of a well: By a part of a well we mean a maximal set of activities of the same well scheduled consecutively in the same resource. Medium granularity. Satisfies the *Same Oil Derrick for a Well* constraint.

In our implementation we chose the last option. Hence, the local search algorithm tries to exchange all pairs of parts of wells, no matter on what resource they were scheduled. That neighborhood is of size $O(n^2)$, where n is the number of parts of wells. On the average, real instances consist of about 22 wells, with an average of 6 activities each well, for a total of 132 activities. The number of parts of wells, on the average, is about 7/6 of the number of wells, which gives around 26 parts of wells. The number of exchanges would then be of the order of $(26 \cdot 25)/2$, or 325. If we would had taken activities as our elements, the number of exchanges would be of the order of $(132 \cdot 131)/2$, or 8646, *i.e.*, almost 27 times bigger. The local search ends when there are no more exchanges that can improve the current solution.

To fully specify the local search algorithm we need a procedure that defines how the neighborhood is searched and which neighbor solution replaces the current one. In the case of iterative improvement algorithms, this rule is called the *pivoting rule* [51], and examples of it are the *first improvement rule* (FIR) and the *best improvement rule* (BIR). With the former one, the algorithm moves to a neighboring solution as soon as it finds a better solution, while with the latter all neighbors are checked and the best one is chosen. In either case, the worst case running time of each iteration is bounded by $O(n^2)$, where n is the number of elements of the neighborhood. In the next section we present a comparison between the FIR and BIR strategies, among other analysis.

5 Computational Results

In this section, computational results for the GRASPS implementation are given, and they are also compared with results obtained with the ORCA implementation, over the same real instances. All tests were run on a plataform equipped with a Sun SPARC Ultra 60 processor running a Solaris 9 operating system at 450 MHz and with 1024 MB of RAM. Both GRASPW and ORCA were allowed to run for 1800 seconds on each instance.

5.1 Typical Instances

Usually, the real instances provided by Petrobras for the WDP consist of 22 wells, with 6 activities each, summing up 132 activities. The number of wells in an instance ranges from 17 to 29 and the number of activities of a well ranges from 2 to 12. Hence, the total number of activities of an instance may vary between 34 and 358 activities. The number of parts of wells, on the average, is 7/6 of the number of wells, which gives around 26 parts of wells in an instance, on the average.

The resources are represented by boats and oil derricks. The number of boats in an instance ranges from 1 to 2, while the number of oil derricks ranges from 2 to 3. Hence, the number of resources in an instance may vary between 3 and 5.

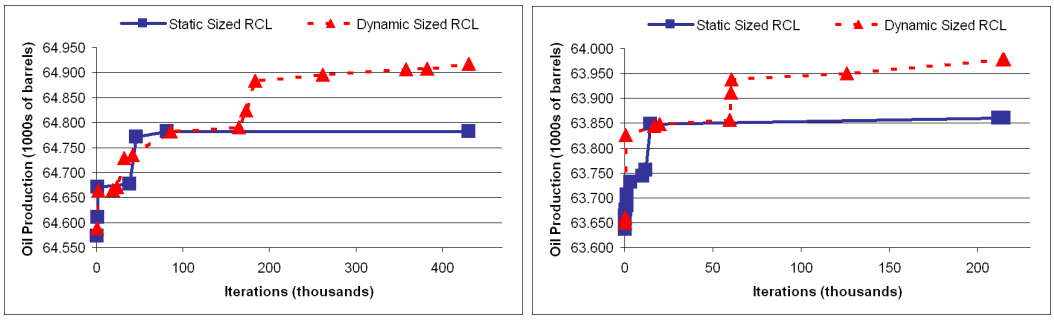
The time horizon of the schedule is between a thousand to three thousand days.

To submit the implementations to tests, we chose 12 distinct instances. In order to reduce the amount of time spent in testing, in some experiments we used only 7 of these instances. We also chose 9 distinct instances where *User Defined Sequences* constraints (see Section 2) were not empty. As GRASP uses randomization in its algorithm, each instance was tested many times, and a medium of the runs were considered in the analysis.

5.2 Setting GRASPS Parameters

In Subsection 4.2 we presented the idea of a *dynamic sized RCL*, in which the number of candidates varies through time. There are two ways we could exploit the dynamic sized RCL: one in which we decrease the number of candidates through time, so that after a broader search, it would focus into a greedier heuristic; and other in which we increase the number of candidates through time, in order to drive away from a local optimum into new regions of the search tree. In the first case, the initial RCL size is set to $max(13, w)$, w being the number of wells, and is decreased by one every 300 seconds without improvement. In the second case, we start with $max(5, w)$ for the initial RCL size and increase it at every 300 seconds without improvement. The first approach did not yield good results when applied to the WDP, generating the same or worse solutions than those found by GRASPS with a static sized RCL. However, the second approach proved promissing. In Figure 5 we depict the algorithm behavior over two real instances of the WDP. Figure 5.a shows the algorithm with dynamic sized RCL generating better solutions after 150 thousand iterations, when the RCL is increased. The same happens in Figure 5.b after 50 thousand iterations. Amongst twelve scenarios tested, four had better solutions with the dynamic sized RCL, totalizing an increase in oil production of around 261 thousand barrels of oil. In the other eight scenarios,

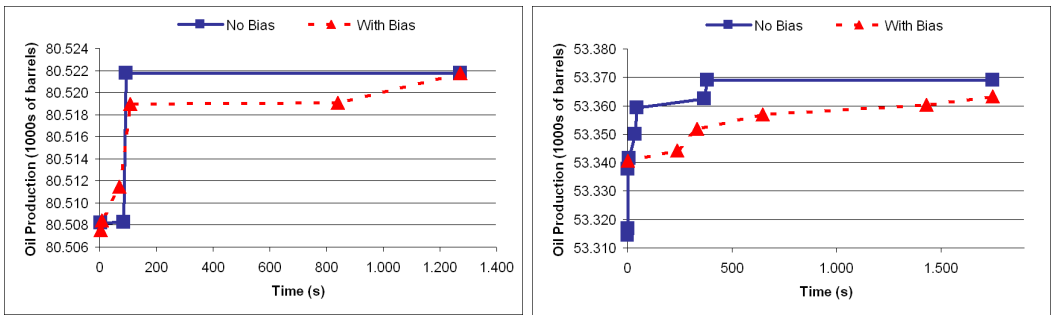
the same solutions were found.



(a) Instance 1 (b) Instance 2

Figure 5: Static Sized RCL x Dynamic Sized RCL.

Another technique considered was to bias the selection towards some particular candidates, those with the highest oil yield, but this strategy did not produce good results. Amongst twelve instances tested, three had slightly worse solutions with such a bias function, totalizing a decrease of 40 thousand barrels of oil. Worse, with the bias function in place, the algorithm takes much more time to find the same solution than when no bias is used. Summing up all the differences in time of the twelve scenarios, we have that with the bias function the algorithm took 3431 more seconds to reach the same results. That is an average of 72% more time. Figure 6 shows that in two instances it took more than twice the time: 1269% (Figure 6.a) and 355% (Figure 6.b).

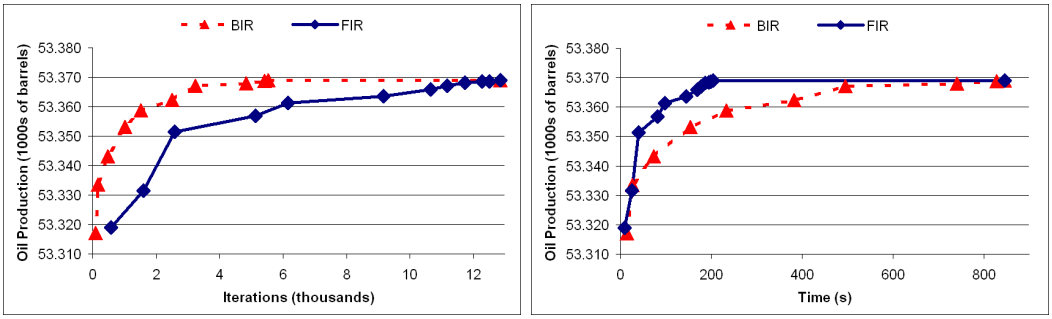


(a) Instance 1 (b) Instance 2

Figure 6: No Bias Function x With Bias Function.

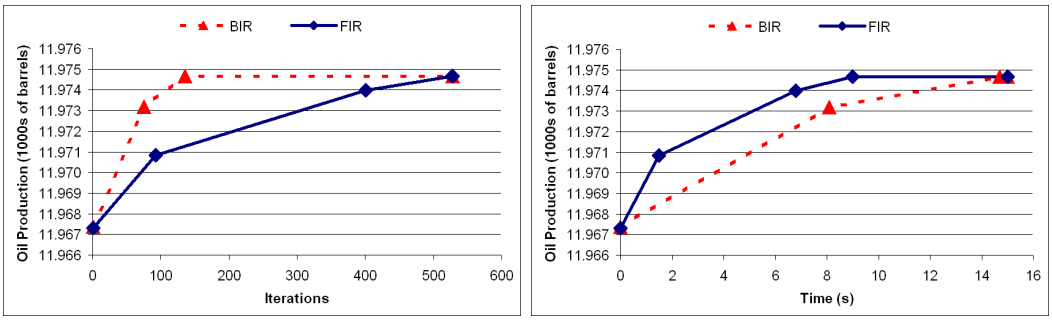
We considered two options for searching the neighborhood and selecting a new neighbor: the *first improvement rule* (FIR) and the *best improvement rule* (BIR). We tested both of them in seven instances. The BIR proved to be better than the FIR when finding

solutions whose production was equal to a predefined target value with the least number of iterations (see Figure 7.a and Figure 8.a). On the average, to find a solution with a predefined production, the BIR strategy used about 60% of the number of iterations of the FIR strategy. On the other hand, the FIR strategy was faster in most instances (see Figure 7.b and Figure 8.b). On the average, to find a solution with a predefined production, the FIR strategy used 66% of the time used by the BIR strategy. On the average, a FIR iteration was almost seven times faster than a BIR iteration. Since to users running time was deemed important, the FIR strategy was found to better suit this problem.



(a) (b)

Figure 7: BIR x FIR: Instance 1.



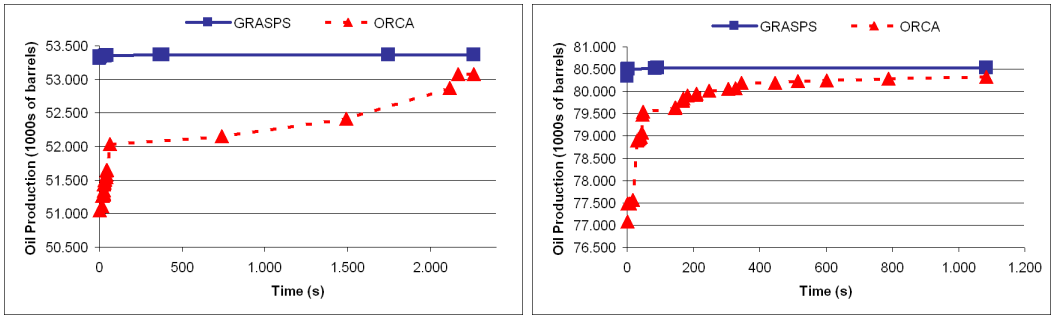
(a) (b)

Figure 8: BIR x FIR: Instance 2.

5.3 Comparative Tests between the ORCA and the GRASPS Implementations

Our GRASPS implementation found better solutions than the ORCA implementation in all eleven real instances tested, and in one it found the same solution. GRASPS achieved 0,14% more oil production on the average which, despite being a small percentage, means an increase of almost one million barrels of oil in total. In fact, the highest gain with

GRASPS was certainly in running time. It found solutions with the same production of those generated by ORCA, on the average, in only 36,3% of the time used by ORCA on the same instances. Figure 9.a shows that the best GRASPS solution has a production 290 thousand barrels of oil higher than the best ORCA solution. Furthermore, GRASPS found a solution with the same oil production of the best ORCA solution within the first second, while ORCA found it only after the 2200 seconds. Similarly, Figure 9.b shows that the best GRASPS solution has a production of almost 200 thousand barrels of oil higher than the best ORCA solution. Moreover, again GRASPS found a solution with the same oil yield as the best ORCA solution within the first second, while ORCA found it only after 1000 seconds.



(a) Instance 1 (b) Instance 2

Figure 9: ORCA x GRASPS.

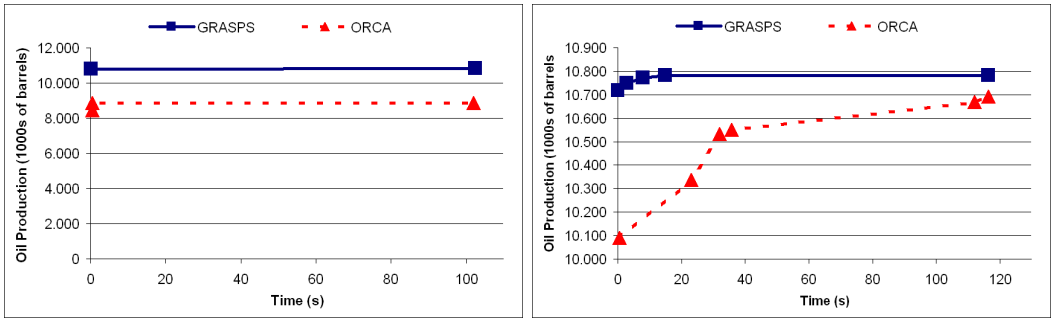
In nine tested scenarios where the *User Defined Sequences* constraints (see Section 2) were not empty, GRASPS found better solutions than ORCA in all of them. In one of them, ORCA could not find any solution, while GRASPS found one with about 26 million barrels of oil production. Comparing the other eight scenarios, GRASPS achieved 5,3% more oil production, on the average, which means an increase of around 4600 thousand barrels of oil, in total. In Figures 10 and 11 we present the results over four such instances. In all of them GRASPS was more effective than ORCA in finding solutions with the highest oil production.

In all accounts, GRASPS has proved to be a better algorithm than ORCA.

6 Conclusion

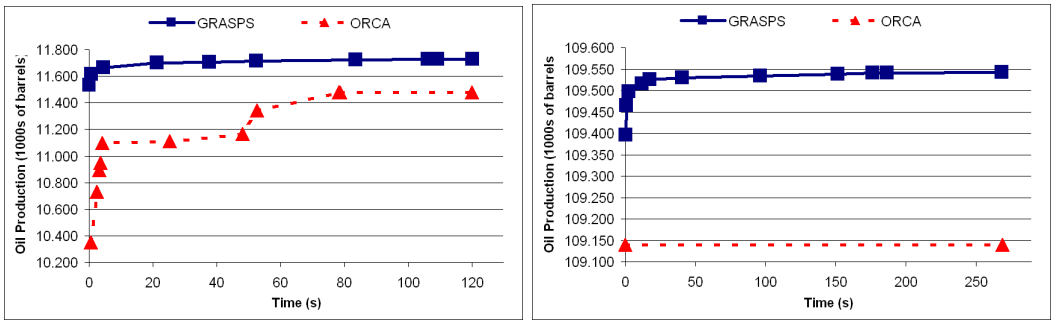
Solving scheduling problems efficiently is of paramount importance to the industry, in general. Petrobras, one of the leading companies on deep seas oil exploration, presented us the oil well drilling problem (WDP). In this paper, we contrast two approaches to the WDP: a constraint programming approach, named ORCA, with a GRASP implementation, dubbed GRASPS.

Computational experiments were conducted on several real instances of the problem. We concluded that GRASPS generates solutions with higher oil production than ORCA.



(a) Instance 3 (b) Instance 4

Figure 10: ORCA x GRASPS.



(a) Instance 5 (b) Instance 6

Figure 11: ORCA x GRASPS.

The results also prove that GRASPS outperforms ORCA, in the sense that it generates solutions with the same oil production in less time than ORCA.

It is worth mentioning that ORCA is built over the ILOG Constraint Programming suite, which consists of sophisticated constraint programming libraries, with more than a decade of development[6]. Moreover, this suite is highly expensive, costing Petrobras hundreds of thousands dollars to acquire it, plus dozens of thousands dollars of maintenance per year. Therefore, using GRASPS, Petrobras may save these costs. Note, on the other hand, that the ILOG suite favours easiness of development, of maintenance and of understanding of the source code, in opposition to the GRASPS implementation, whose modifications demand much more effort to implement. Note also that ORCA results surpassed manual results.

[6]See http://www.ilog.com

References

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

[2] J.F. Bard, T.A. Feo, and S. Holland. A grasp for scheduling printed wiring board assembly. *I.I.E. Transactions*, 28:155–165, 1996.

[3] S. Binato, W. J. Hery, D. Loewenstern, and M. G. C. Resende. A greedy randomized adaptive search procedure for job shop scheduling. In P. Hansen and C. C. Ribeiro, editors, *Essays and surveys on metaheuristics*. Kluwer Academic Publishers, 2001.

[4] S. Binato and G. Oliveira. A reactive grasp for transmission network expansion planning. *Essays And Surveys In Metaheuristics - Kluwer Academic Publishers*, 2002.

[5] B. Bullnheimer, R. F. Hartl, and C. Strauss. Applying the ant system to the vehicle routing problem. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, *Metaheuristics: Advances and Trends in Local Search for Optimization*, pages 285–296, Boston, 1999. Kluwer Academic Publishers.

[6] W. Clocksin. Principles of the delphi parallel inference machine. *Computer Journal*, 30(5):386–392, 1987.

[7] A. Colorni, M. Dorigo, and V. Maniezzo. Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics, and Computer Science*, 1(34):39–53, 1994.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, September 2001.

[9] Juliana Martins do Nascimento. Hybrid computational tools for the optimization of the production of petroleum in deep waters. Master's thesis, Institute of Computing, University of Campinas, 2002.

[10] Marco Dorigo, Gianni Di Caro, and Luca M. Gambardella. Ant algorithms for discrete otimization. Technical report, Universite Libre de Bruxelles, 1999.

[11] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system: Optimization by a colony of cooperating agents. *IEE Transactions on Systems, Man and Cybernetics*, 26(1):1–13, 1996.

[12] H. Faria, S. Binato, and M. Resende. Power transmission network design by a greedy randomized adaptive path relinking approach. *Proceeding of the Fourth Metaheuristic International Conference*, 2001.

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

[14] 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.

[15] 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.

[16] T. A. Feo, K. Venkatraman, and J. F. Bard. A grasp for a dificult single machine scheduling problem. *Computers & Operations Research*, 18:635–643, 1991.

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

[18] P. Festa and M. Resende. Grasp: An annotated bibliography. *Essays And Surveys In Metaheuristics - Kluwer Academic Publishers*, 2002.

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

[20] Fred Glover. Tabu search - part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[21] Fred Glover. Tabu search - part ii. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[22] Fred Glover. Tabu search: a tutorial. *Interfaces*, 20(4):74– 94, July-August 1990.

[23] Fred Glover and Manuel Laguna. Tabu search. In C. R. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, chapter 3, pages 70–150. McGraw-Hill, 1995.

[24] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, Massachusetts, 1997.

[25] Fred Glover and Manuel Laguna. Tabu search. Taken from the web site: `www-bus.colorado.edu`, `faculty/laguna/Papers/ts.pdf`., August 2000.

[26] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1:607–613, August 1995.

[27] G. Hasle, R.C. Haut, B.S. Johansen, and T.S. Ølberg. – an application of constraint reasoning. In *Proceedings of PACT'97 (The 1997 International Conference on Parallel Architectures and Compilation Techniques)*, San Francisco, Callifornia, November 1997.

[28] ILOG. *ILOG Solver 4.4 Reference Manual*. ILOG, June 1999.

[29] J. Jaffar and J.L. Lassez. Constraint logic programming. In *14^{th} ACM Symposium on Principles of Programming Languages*, pages 111–119, Munique, Alemanha, 1987.

[30] Manuel Laguna. Tabu search tutorial. II Escuela de Verano Latino-Americana de Investigacion Operativa, Mendes, RJ, 1995.

[31] Manuel Laguna and J. L. Gonzalez-Velarde. A search heuristic for just-in-time scheduling in parallel machines. *J. of Intelligent Manufacturing*, 2:253–260, 1991.

[32] C. Le Pape. Implementation of resource constraints in ILOG SCHEDULE: a library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering*, 3(2), 1994.

[33] Jonathan Lever, Mark Wallace, and Bary Richards. Constraint logic programming for scheduling and planning. *BT Technology Journal*, 13(1), 1995.

[34] H. Ramalhinho Loureno, J.P. Paixo, and R. Portugal. Metaheuristics for the bus-driver scheduling problem. In *Economics Working Papers*, volume 304. Department of Economics and Business, Universitat Pompeu Fabra, 1998.

[35] Udi Manber. *Introduction to Algorithms: A Creative Approach*. Addison Wesley, January 1989.

[36] Kim Marriott and Peter J. Stuckey. *Programming with Constraints: An introduction*. MIT Press, Cambridge, Massachusetts, 1998.

[37] Pedro Meseguer. Interleaved depth-first search. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2:1382–1387, August 1997.

[38] N. J. Nilsson. *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill, 1971.

[39] Mauricio G. C. Resende and Paola Festa. An updated bibliography of grasp. Technical Report TD-5SB7BK., AT&T Labs, October 2003.

[40] Mauricio G. C. Resende and Celso C. Ribeiro. Greedy randomized adaptive search procedure. *AT&T Labs Research Technical Report TD53RSJY*, 2002.

[41] M. Ribeiro. A grasp for job shop scheduling. *Essays And Surveys In Metaheuristics - Kluwer Academic Publisher*, 2002.

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

[43] R.Z. Rios-Mercado and J.F. Bard. Heuristics for the flow line problem with setup costs. *European J. of Operational Research*, 110(1):76–98, 1998.

[44] L. I. D. Rivera. Evaluation of parallel implementations of heuristics for the course scheduling problem. Master's thesis, Instituto Tecnologico y de Estudios Superiores de Monterrey, Monterrey, Mexico, 1998.

[45] B. Roy and B. Sussmann. Les problmes d'ordonnancement avec contraintes disjonctives. In *Note DS No 9 bis*, Paris, 1964. SEMA.

[46] M. Souza, C. Duhamel, and C. Ribeiro. A grasp heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy. *Kluwer Academic Publishers*, 2002.

[47] A. Srinivasan, K. Ramakrishnan, and K. Kumaran. Optimal design of signaling networks for internet telephony. *Lucent Technologies, Labs Innovations*, 2000.

[48] Toby Walsh. Depth-bounded discrepancy search. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2:1388–1393, August 1997.

[49] J. Xu and S. Chiu. Solving a real-world field technician scheduling problem. *In Proceedings of the International Conference on Management Science and the Economic Development of China*, pages 240–248, July 1996.

[50] J. Xu and S. Chiu. Effective heuristic procedure for a field technician scheduling problem. Technical report, US WEST Advanced Technologies, Boulder, CO, USA, 1998.

[51] M. Yannakakis. Computational complexity. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 19–55, Chichester, 1997. John Wiley & Sons.

[52] T. H. Yunes, A. V. Moura, and C. C. de Souza. A hybrid approach for solving large scale crew scheduling problems. In *Lecture Notes in Computer Science,* vol. 1753, pages 293–307, Boston, MA, EUA, January 2000. Anais do *Second International Workshop on Practical Aspects of Declarative Languages (PADL'00).*

[53] T. H. Yunes, A. V. Moura, and C. C. de Souza. Solving very large crew scheduling problems to optimality. In *14^{th} ACM Symposium on Applied Computing (SAC'00)*, Como, Itlia, March 2000.