GRASP STRATEGIES FOR SCHEDULING ACTIVITIES AT OIL WELLS WITH RESOURCE DISPLACEMENT

Arnaldo V. Moura[1], Romulo A. Pereira[2], and Cid C. de Souza[3]

Institute of Computing, University of Campinas

[1] Instituto de Computao, UNICAMP, Caixa Postal 6176
13084-971 Campinas, SP
arnaldo@ic.unicamp.br
+55 (19) 3788-5859

[2] romulo_a_pereira@yahoo.com.br

[3] cid@ic.unicamp.br

Abstract

Before promising locations at petroliferous basins become productive oil wells, it is necessary to complete development activities at these locations. The scheduling of such activities must satisfy several conflicting constraints and attain a number of goals. Moreover, resource displacements between wells are also important. We describe a Greedy Randomized Adaptive Search Procedure (GRASP) for the scheduling of oil well development activities with resource displacement. The results are compared with schedules produced by a well accepted constraint programming implementation developed by a large oil company. Computational experiments over real instances proved that the GRASP heuristics implementation was competitive, outperforming the constraint programming implementation.

Key-words: *Offshore petroleum, GRASP, constraint programming, scheduling.*

Oil and gas are important fossil ingredients for making plastics, dyes, kerosene, gasoline and many other products. A significant amount of these fossil fuels is extracted from oceanic basins, *e.g.*, from the offshore Marlin basin in Rio de Janeiro, Brazil. Petrobras is a company with recognized expertise in oil exploration in deep sea waters, being also one of the twenty biggest oil companies in the world. Usually, Petrobras explores diverse petroliferous basins, each with hundreds of promising spots where productive oil wells could be located. However, before these places are turned into productive wells they must be developed, that is, a sequence of engineering activities must be completed at each promising spot, to render them ready for oil extraction. Oil derricks and ships are used to complete these activities. These resources have to move from one spot to another and such displacements must be considered when sequencing the activities. Furthermore, such resources are limited and expensive, either in acquisition or rent value, and must be used efficiently.

More specifically, when a spot is considered a promising oil well, oil derricks are sent there to accomplish the due *drilling* operations. After a well is drilled, the preparation for oil extraction develops in several stages. First, in the stage called *completion*, oil derricks place Wet Christmas Trees (or WCTs, structures where hydraulic valves are attached) at the mouth of the wells in order to avoid oil leakage. Later, boats connect pipelines between WCTs and manifolds, this stage being called *interconnection*. Manifolds are metallic structures installed by boats at the sea floor. Their use prevents the need for exclusive pipelines connecting each well to the surface, which would be prohibitively expensive. Once this stage is completed, oil extraction can begin. For that, Stationary Units of Production (SUPs) are anchored at specific locations in the surface and boats interconnect manifolds to them. SUPs are used to process, and possibly store, the extracted products. Later, ships fetch the products from SUPs to land storage sites or other processing units. If the oil outflow is very high or a SUP does not have storage capacity, a petroliferous platform may be installed at the surface.

Usually, there are two types of wells that may be developed: (i) *productive wells*, which are those that have an oil yield; and (ii) *injection wells*, where only maintenance activities are executed and oil is not extracted from them.

The oil *well development with resource displacement problem* (WDRDP) can be summarized thus: given a set of promising spots, the activities to be executed at each location, and the available resources, find a scheduling of the activities and resources, fulfilling several conflicting engineering and operational constraints, including the displacements of the resources, in such a way as to optimize some objective criteria. In this work, the specific WDRDP faced by Petrobras is studied. This WDRDP imposes much more realistic constraints than other similar studies (do Nascimento (2002), Pereira et al. (2005)). Actually, Pereira et al. (2005) is an early version of this paper, which did not consider resource displacement. Here, we take this characteristic into account when describing our GRASP heuristic strategies for maximizing oil production within a given time horizon. In addition, we discuss other advanced techniques not explored in Pereira et al. (2005).

A project team from Petrobras addressed the same problem and developed a Constraint Satisfaction (cf., Marriott & Stuckey (1998)) model using ILOG's Solver and Scheduler[1] (ILOG (1999)). After four years of development and testing, the tool, named

[1]Registered trade marks of ILOG Inc.

ORCA[2], became operational and very successful (Petrobras (1999)). Nowadays, the ORCA solver is often used by engineers both to define a good scheduling for the drilling activities and, also, to analyze the need for acquiring or renting new resources. They confirmed that ORCA generates better solutions than those made by humans. In one real instance, 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, an expenditure of US$ 15 million was avoided, while anticipating oil production by 26 days. Despite the good performance of ORCA, searching for even better solutions is still important, since a tenth of a percent improvement in the oil production may represent an increase of millions of dollars in the company's revenue.

Before we proceed, we should emphasize that Petrobras allowed us to use only ORCA's executable files. We had no access to the source code. Therefore, we are unaware of ORCA's inner details, other from some general ideas that were cleared to go public. This information is spread throughout this text. In light of that, this work should be viewed as a successful application of a GRASP heuristics to a relevant real-world problem. The comparisons we make between the GRASP heuristics and ORCA's constraint programming strategies is not to be construed upon in order to establish the superiority of one technique over the other. Moreover, that would not be a fair comparison, since there is certainly some room for improvement in ORCA's code that we could not exploit, as explained above. Nevertheless, since the technicians from Petrobras are very confident about the quality of ORCA's solutions, it is legitimate for us to use them as a measure to assess the performance of our GRASP implementation.

Section 2 discusses GRASP implementations for the WDRDP. Section 3 presents our computational results and compares them to other results derived from the constraint programming implementation presently running at Petrobras. Finally, some concluding remarks are offered in the last section.

1 The Well Development Problem

As already mentioned, here we will deal with a new facet of the problem, the resource displacement. This constraint forces a period of inactivity for the resource, so that it can unanchor, travel between two wells and anchor in the new spot. We call this period the set-up time. It is present between activities of different wells which are scheduled consecutively. In Figure 9.a we see a typical schedule with no resource displacement. In the figure, the light gray rectangles are activities, or group of activities associated with a same well. The dark gray rectangles represent resources. Each resource executes the sequence of activities that appear in the corresponding horizontal line. In Figure 9.b, we present a schedule with resource displacement. Note that between each pair of wells there is a void representing the set-up time.

The set-up time is a constant for each resource. It is calculated by engineers, based on the average speed of the resource and on the average distance between spots in the field, plus the average time to anchor and unanchor. As engineers noted when scheduling the activities at wells of an oil basin, the average distance between these wells does not differ much from the average distance traveled by the resources. The weather, which influences

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

the speed of the resource, was also considered in the average speed.

However, different resources may have different set-up times. Consider, for example, resources X and Y in Figure 9.b. Note that the set-up time of Y is twice longer than the set-up time of X. This may alter the selection of which resource will execute activities at a well. For example, wells $X2$ and $Y2$ have their "end of execution" activities scheduled at the same instant. This would not be the case if both resources X and Y had the same set-up times. When resource displacement is being considered, a good criterion to select which resource will execute which activity may be even more important in the search process, given that the set-up times may postpone the schedule of activities.

According to its developers, ORCA faced difficulties when treating the resource displacement constraints. Several constraint programming techniques available in the ILOG Solver and Scheduler, like the usage of transition times, were tried by the Petrobras team when dealing with these constraints. A first approach created virtual activities to represent the set-up time. But, since it is not known in advance if an activity from another well will be scheduled consecutively, it would be necessary to create virtual activities dynamically during the search for solutions. Of course, they would have to be destroyed when backtracking. This was a complex process and slowed down the search considerably. Instead, an *if-then constraint* was set, imposing that if the previous activity executed by the resource was from another well, then the current activity was shifted by the corresponding set-up time. As this constraint was imposed between each pair of activities, the number of constraints to be added to the model was quadratic in the number of activities. The effect was a slow down in the execution of ORCA, causing it to generate poor solutions for many instances. As a general remark, the use of the readily available techniques to treat the resource displacement constraints slowed down the execution of ORCA, reducing its ability to generate good solutions in an satisfactory amount of time.

1.1 The WDRDP Constraints

Any solution for the WDRDP must obey the following constraints involving the oil well development activities:

C1. *Technological Precedence*: sets an order between pairs of certain activities. When considering the precedence between the start and finish of the activities in each pair, any of the four possible combinations can be present.

C2. *Mark-Activity*: an activity must finish before or initiate after a fixed date, or *mark*, with or without lag time. This date is often related to some external event, *e.g.*, the installation of a petroliferous platform.

C3. *Baseline*: sets the start date of the activities.

C4. *Use of Resources*: to execute an activity, due to its intrinsic nature, the resource used must match some operational characteristics. For a boat, it must be verified if the on-board equipments can operate at the specified depth. For an oil derrick, its type and capabilities must be verified, as well as its maximum and minimum depth of operation and drilling. All resources are disjunctive, that is, an activity can require at most one resource at any given time.

C5. *Concurrence*: two activities at the same well, or executed by the same resource, can not be simultaneous.

C6. *Unavailability*: resources may be unavailable for a period of time, either for maintenance reasons or due to contract expiration.

C7. *User Defined Sequences*: the user can specify a sequence for the drilling activities or for the "start production" activities of different wells, depending on the type of this sequence. These sequences are specified by engineers in order to avoid loss of pressure in the oil field. If a well A appears before well B in the sequence, then the activity of well A must terminate before the start of activity of well B can be scheduled.

C8. *Surface Constraints*: represented by a polygonal security area defined around a well. When the center of a well is inside the restricted area of another well, activities executed at both wells cannot be simultaneous. These constraints must be verified between pairs of mobile and pairs of mobile and anchored oil derricks.

C9. *Cluster Constraints*: an activity may be part of a cluster, which is a set of activities that must use the same resource.

C10. *Oil Derrick Displacements*: when an oil derrick moves between two wells, a set-up time will be considered. Therefore, unnecessary displacements must be avoided, for example, by making the same oil derrick execute as much activities at a well as possible. For more details, see Section 1.

The oil yield is calculated as follows. Each well has an associated outflow and an activity that marks the beginning of its production. When this last activity is concluded, the well is considered in production. The yield is obtained by multiplying the oil outflow by the period between that instant and the established time horizon of production. If the start production activity is set for a time after the horizon, the corresponding yield is disregarded. The objective is to obtain a scheduling of all activities, satisfying all constraints, while maximizing the oil yield.

Other goals to be attained by automating the scheduling of the activities are:

1. *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, modifications in already committed plans would not result in new hours, or days, spent in rescheduling.

2. *Better resource allocation.* With an automated scheduling, all highly skilled engineers responsible for the manual scheduling can receive other duties.

3. *Savings.* The automated schedule will, usually, result in a better use of boats and derricks, thereby saving considerable operational costs.

As can be seen, the WDRDP is a difficult combinatorial optimization problem. In fact, it is simple to devise a polynomial-time reduction from the classical *Job Shop Scheduling Problem (JSP)* to the WDRDP, thus showing that the WDRDP is NP-hard.

2 GRASP Strategies for the WDRDP

Despite its conceptual simplicity, GRASP (Feo & Resende (1995)) is a well studied metaheuristics which has been successfully applied to a wide variety of optimization problems (Glover & Laguna (1997), Festa & Resende (2002)). In particular, GRASP implementations have dealt well with scheduling problems (Bard & Feo (1989), Feo & Bard (1989), Feo et al. (1995), Bard et al. (1996), Binato et al. (2002)). After some initial investigation (do Nascimento (2002)), a GRASP approach seemed appropriate to solve the WDRDP. In fact, a GRASP heuristics had already proved itself valuable in dealing with a simpler version of the WDRDP, where resource displacement was not considered (Pereira et al. (2005)).

Subsection 2.1 reviews some GRASP basics, while subsection 2.2 presents some more advanced GRASP techniques. The remaining subsections 2.3–2.5 describe our specific implementation, named GRASP-WDRDP (or GRASPW, for short), which was designed to solve the full WDRDP, with resource displacement.

2.1 Greedy Randomized Adaptive Search Procedure (GRASP)

In the GRASP methodology, each iteration consists of two phases: a *construction phase* followed by a *local search phase* (Feo & Resende (1995)). See Figure 1.

In the construction phase, a feasible solution is built one element at a time. See Figure 2. At each step of the construction phase a candidate list is obtained by ordering all elements that can be added to the solution at this point. The ordering follows a greedy function that measures the, maybe myopic, benefit of inserting each candidate element into the partial solution being constructed. After the candidate list is constructed, an initial segment is selected, forming the *Restricted Candidate List* (RCL). The construction step is completed by randomly selecting an element from the RCL and effectively inserting it into the partial solution. The construction cycle repeats itself until a complete solution is obtained. The adaptive component of the heuristic arises from the fact that the benefits computed by the greedy function are updated at each step, in order to reflect the changes brought on by the selection of the candidate in the previous step. The probabilistic component stems from the random choice of one of the candidates in the RCL, not necessarily the best one, to be added to the partial solution. This mechanism allows for different solutions to be obtained at each iteration, while not necessarily jeopardizing the adaptive greedy component.

The solutions generated by the construction phase are not guaranteed to be locally optimal. Hence, in the local search phase, a local search procedure is applied to improve the constructed solution. The search phase is a standard deterministic local search algorithm that seeks to optimize the solution built in the previous construction phase.

2.2 GRASP Advanced Techniques

As the problem increased in complexity, compared to the problem with no resource displacement studied in Pereira et al. (2005), we decided to consider some improvements and alternative techniques to be introduced in the basic GRASP procedure. They are discussed in the sequel.

Bias Function: 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, with the elements of the RCL being assigned equal probabilities. However, any probability distribution can be used to bias the selection toward particular candidates. Bresina (1996) proposed a construction mechanism based on the *rank* $r(\alpha)$ assigned to each candidate element α, according to its value, measured by the greedy function, $v(\alpha)$. However, as the sorting of elements to obtain the *rank* has time complexity $O(n \log n)$, where n is the number of elements in the RCL, and since the sorting must be repeated at each choice of an element, this process could slow down the implementation. Thus, we decided to use directly the value $v(\alpha)$ of the candidates to create the following bias functions:

- uniform: $bias(\alpha) = 1$
- linear: $bias(\alpha) = v(\alpha)$
- log: $bias(\alpha) = \ln v(\alpha)$
- exponential: $bias(\alpha) = e^{v(\alpha)}$
- quadratic: $bias(\alpha) = v(\alpha)^2$
- square root: $bias(\alpha) = \sqrt{v(\alpha)}$

Once the value of the bias function is evaluated for all elements of the RCL, the probability of the candidate α being chosen is:

$$\frac{bias(\alpha)}{\sum_{\alpha' \in RCL} bias(\alpha')}$$

The bias function was inserted into the new construction phase, see Figure 4.

Proximate Optimality Principle (POP): This technique is based in the idea that "good solutions at one level are likely to be found 'close to' good solutions at an adjacent level" (Glover & Laguna (1997)). Fleurent & Glover (1999) provided a GRASP interpretation of this principle. They suggested that imperfections introduced during steps of the GRASP construction phase can be removed by applying local search during (and not only at the end of) the GRASP construction phase. A practical use of the POP in a GRASP implementation would be to apply a local search during a few intermediate points in the construction phase, and not at the end of each construction iteration. Binato et al. (2002), when dealing with a JSP, applied a local search when 40% and 80% of the construction moves had been taken, as well as at the end of the construction phase. We tested this idea by running a local search procedure at several intermediate points during the construction phase. The best results were obtained when the same 40% and 80% of the construction moves had been taken. Given the closeness between the JSP and the WDRDP (see Section 1.1), this was not surprising.

The local search strategy we used in order to test the POP was a usual well swapping process, with the proviso that we only swap wells that have already been scheduled. A pseudo-code for the implementation of POP in the new GRASPW procedure is illustrated in Figure 5. The routine is activated during the construction of a greedy solution, see Figure 4.

One possible shortcoming of the basic GRASP method is the independence of its iterations, *i.e.*, the fact that one iteration does not learn from the history of solutions found in previous iterations. This is so because the standard algorithm discards information about any solution encountered that does not improve the incumbent solution. But it is known that information gathered from good solutions can be used to implement memory-based procedures. We show some of these strategies below. There are, of course, a number of other possibilities to induce dependence among constructive phases of GRASP, such as ant colony optimization.

Intensification: Fleurent & Glover (1999) proposed a scheme to introduce long term memory in a GRASP heuristics using a set of elite solutions.

In our case, in order to be included in this elite set, a solution must be better than all elite solutions already in the set, according to their objective function values. Or, else, it must be better than the worst solution of the set, while still being sufficiently different from all elite solutions in the set. A solution is considered sufficiently different from another one if the number of activities that have different start dates in both, or that have a different resource allocated to both, is higher than the number of activities divided by the number of resources. As the number of resources is usually smaller than five, two solutions will be sufficiently different if at least 20% of their activities have a different start date, or a different resource allocated to both. We did test other thresholds values that could be used to ascertain when two solutions were considered different. This was done during the parameter setting phase in the development of our application. The value we retained proved to be the best among all other thresholds tested.

The size of the elite set, fixed at 5 elements, was also obtained by testing different values over real instances. A five element set produced the best results, while keeping the elite set quite small. At the start, the first 5 solutions populate the elite set. From this point on, an element is always removed from the elite set each time we insert a new one. We tried two strategies for removing elements from the set. In the first one, we removed the element with the smaller oil production, among all elements in the elite set. In the second trial, we removed the element with fewer differences in its scheduling, when compared with the new element. Again, tests run on real instances proved that the first approach gave superior results.

Next, we show how the elite solutions biased the selection of the candidates in the GRASP construction phase. To each candidate α we evaluate an intensity function, $Int(\alpha)$, defined as:

$$Int(\alpha) = \frac{\sum_{e_i \in S} Prod(e_i)}{Max_{e_j \in E} Prod(e_j)},$$

where S is the set of elite solutions in which the element α has the same start date and resource allocated to as the solution being built, $Prod(\ )$ is the oil yield associated with a solution, E is the set of elite solutions, and Max returns the maximum value in the indicated set. With the size of the elite set limited to 5 elements, the computation of $Int(\alpha)$ was very fast.

Let *iter* be the number of iterations executed and let k be a parameter. We use the intensity function to define the bias function, $biasInt(\alpha)$, as:

$$biasInt(\alpha) = bias(\alpha) + Int(\alpha) \times (iter/k).$$

The fraction $iter/k$ is used to reinforce the role of the intensity function as the number of iterations increases and, thus, the quality of the elite solutions in which the intensity function is based also possibly increases.

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

The use of path-relinking within a GRASP procedure was first proposed in Glover et al. (2000). It was followed by several extensions, improvements and successful applications (Aiex et al. (2005), Canuto et al. (2001), Resende & Ribeiro (2003)). Two basic strategies are:

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

According to Resende & Ribeiro (2002) the second strategy is more promising. Another consideration, also from Resende & Ribeiro (2002), is that exploring the two trajectories, in both directions, between two solutions results in small gains and demands twice the time. Thus, it is usually more appropriate to explore only one path, the one from the best solution to the other, as better solutions are more often found near the neighborhood of the best solutions (Resende & Ribeiro (2002)).

An illustration, in pseudo-code, for the path-relinking strategy appears in Figure 6.

Selective Local Search (SLS): Another approach that can be used within the standard GRASP procedure is to apply local search only in those solutions sufficiently different from the elite solutions, or that have a good quality, *i.e.*, better than the worst of the elite solutions. In other words, we apply local search only when there are higher chances of obtaining better solutions, improving the efficiency of the algorithm. This idea was also used in our new implementation of the basic GRASP procedure, see Figures 3 and 7.

2.3 The New GRASP Solver Implementation: GRASPW

The GRASPW implementation was constructed using the C/C++ programming language. Our model uses two types of integer variables. One represents the beginning of execution of each activity in the corresponding well. Values in the domain of these variables represent days, and the range of each domain varies from zero to the time horizon. The latter depends on the instance being treated, but varies usually between 1000 and 3000 days. Of course, during the search for solutions, as activities are scheduled, other activities have the domain size of the corresponding variables diminished. The second type of variables represents which resource will execute each activity in a well. Their domains are characterized by a set of the possible resources, of which one must be chosen to execute the corresponding activity.

All the constraints described in Section 1 were enforced. Three constraints, namely, C2, C3 and C4, were set while reading the problem data, before the search begins. Note that, in these cases, all information needed to check the constraints are already present. The other constraints were dealt with during the search for solutions, the variables involved being assigned single values.

Constraint C10, that deals with resource displacement and which was responsible for a loss of performance of the original ORCA search algorithm, was treated here in a simple way. We displace the current activity by at least the amount of the set-up time, if the previous activity scheduled in the same resource is from a different well.

A pseudo-code for the GRASPW implementation appears in Figure 3, and can be compared with Figure 1.

2.4 The Construction Phase

The following two adaptations were made to the procedure illustrated in Figure 1:

- The search procedure was interrupted by a time limit instead of by the number of iterations; and
- During a complete run of the GRASPW meta-heuristic, the value of *ListSize* (see Figure 1) can be monotonically incremented by a fixed amount when a predefined interval of time is reached with no improvement on the best solution. This allows the algorithm to explore larger regions of the search space. Alternatively, during a run of GRASPW, the value of *ListSize* can be monotonically decremented between iterations, thus focusing into a greedier heuristic. With this scheme we obtain a *dynamic sized RCL*, in opposition to the original *static sized RCL*. Note that, as GRASP iterations are independent, one could imagine that there is no difference between increasing and decreasing the RCL size. However, as we do not know in advance the amount of time the algorithm will execute in each run, or when the RCL size will be altered, we can not anticipate the result of a GRASPW run, when increasing or decreasing the RCL size. Our implementation proved that, for the WDRDP, the best strategy was to increase the RCL size, see Subsection 3.2 and Figure 3.

A pseudo-code for the new construction phase implementation is illustrated in Figure 4. As in the ORCA implementation, we seek solutions with the highest oil yield. To this end, the construction phase illustrated in Figure 2 was modified thus:

1. The first time the construction phase is initiated, we set *ListSize* to one, and the algorithm behaves like a greedy heuristic. If there are few constraints obstructing

the greedy heuristic, it tends to generate good or even very good solutions. For example, in six out of seventeen real instances, the best solution was found in the first pass of the construction phase. However, in the other eleven cases the greedy solutions were much poorer than the final corresponding GRASP solutions.

2. 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 are not productive and therefore must not be scheduled before production wells, unless there are technological constraints forcing such a schedule. We schedule the wells and not single activities because otherwise many solutions created would have a huge number of displacements of resources between wells, which would affect the oil production negatively.

3. The evaluation of incremental costs (line 3 of Figure 2) assesses how much oil a well can offer until the end of the time horizon. In order to choose the best wells to enter the RCL, we order them by the value of their oil yields.

   Initially, only the well's own yield is estimated. In order to estimate the production of a well, we simulate the scheduling of this well, i.e., we insert this well in the solution and obtain its production, returning the solution to its original state afterwards. If there are wells which are constrained to succeed the current well, we also estimate the production of these successors. As this process advances, we keep the maximum among all these yields, i.e., the well's own yield and the yields of all its constrained successors. This is the production value used to order the elements that enter the RCL.

   Actually, we tried many other possibilities like, for example, summing up the yield of the well and the yields of all its successors. The best schema we found, according to our tests, was the one that retained the maximum yield.

4. In the construction phase, the next element to be introduced in the solution is chosen uniformly from the candidates in the RCL (line 5 of Figure 2). However, any probability distribution can be used to bias the selection. See Section 2.2 and Figure 4. We tried some bias functions in the selection of the candidates, as will be discussed later.

5. To schedule the candidate well (line 7 of Figure 2), the routine is as follows. See also Figure 8.

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

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

   (b) choose a resource for this activity that can execute it, and that can complete the activity the earliest;

(c) set the start time of the activity at the earliest possible time, *i.e.*, the maximum between the earliest time the resource is available to execute the activity (considering the constraints, including the set-up time) and the minimum start time of the activity; and

(d) all activities that are constrained to succeed the chosen one must have their minimum start times updated so that all constraints are satisfied.

The scheduling of a well is done so as to satisfy all constraints, including the seven ones not yet enforced while reading the problem data. In case of violations, and this can be tested after each activity is scheduled, the construction of this solution is aborted and a new one is started. We could, instead, backtrack a few steps, but this would slow down this phase, especially if the first steps were not appropriate. We note that, in the construction phase, almost all solutions are constructed until they are complete. We used some special strategies to help completion. Specifically, we created a routine that, during the partial construction of solutions, it searches for activities that have a very short domain, and that have not yet been scheduled. If such activities are found, we try to schedule them at this point. We do this to prevent other activities to be scheduled before the problematic ones, thus obliterating the short domains and rendering invalid the solution being constructed. After a well is scheduled, any activities that must succeed it have their minimum start times updated to satisfy all constraints. If that is not possible, the construction of this solution is also aborted.

2.5 The Local Search Phase

As we have no guarantees that the solution found in the construction phase is locally optimal, a local search is used to improve the solution. To this end, an appropriate neighborhood was defined, so as to permit explorations that quickly lead to better solutions. The 2-exchange local search algorithm based on the disjunctive graph model of Roy & Sussmann (1964) was used. The same neighborhood was used in Binato et al. (2002) for a Job Shop Scheduling problem. In order to apply the 2-exchange local search to the WDRDP, we swap two elements in the scheduling. For example, consider the schedule presented in Figure 10.a. Groups of activities (light gray rectangles) are executed by the resource (dark gray rectangles) which is depicted in the same horizontal line. The size of the rectangles represents the execution time of the corresponding elements. We swap elements A and B, which results in the schedule shown in Figure 10.b. Since the execution time of elements A and B can be different, all activities after them may have their start times updated.

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

1. *An activity.* Very small granularity, giving rise to huge neighborhoods (do Nascimento (2002)) and, worse, moving an activity to another position would possibly force its predecessors and successors in the same well to be moved as well, in order to avoid the displacements that may result from constraints of type C10;

2. *A well.* With higher granularity. But since the sequence of activities in a well may be splitted in the present schedule due to constraints of type C1, moving all

activities takes time to verify all constraints, and exchanging the whole well may not be possible even though exchanging only part of it could be;

3. *Part of a well.* That means a maximal set of activities of the same well scheduled consecutively in the same resource. With medium granularity, it avoids the displacements that may result from constraints of type C10.

In our implementation, we chose the last alternative, where the local search algorithm exchanges all pairs of parts of wells, no matter on what resource they have been scheduled. That neighborhood is of size $O(n^2)$, where n is the number of parts of wells. For practical instances, this is one order of magnitude smaller than the neighborhood that uses activities as the moving elements. See Figure 7.

To fully specify the local search phase we need a rule that defines how the neighborhood is searched and which solution replaces the current one. This rule is called the *pivoting rule* (Yannakakis (1997)), and examples of it are the *first improvement rule* (FIR) and the *best improvement rule* (BIR). In the first case, the algorithm moves to a neighboring solution as soon as it finds a better solution; in the second case, 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 in the neighborhood. In the next section we present a comparison between these two alternatives.

Besides the improvements proposed above to the standard GRASP procedure, all the advanced techniques presented in Section 2.2 were implemented and tested.

It is also worth saying a few words about ORCA's innerworkings. The ORCA algorithm uses the basic ILOG Solver and ILOG Scheduler libraries. As usual, after all constraints are imposed, the mechanism of constraint propagation is handled directly by the ILOG libraries. The search procedure is based on backtracking coupled with a branch-and-bound strategy. When traversing the search tree, the Solver always chooses the best open nodes with respect to a given evaluation function, before moving to that node in order to expand it. The evaluation function used in ORCA is the Limited Discrepancy Search (LDS), which was first defined in (Harvey & Ginsberg 1995).

3 Computational Results

In this section, computational results for the GRASPW implementation are discussed. They are also compared with results obtained with the ORCA implementation over the same real instances.

Beforehand, it is important to mention that we treated the ORCA algorithm as a black-box, for comparison purposes. That is, we did not tinker with its code. On the other hand, ORCA is an application that took Petrobras, a very large and competitive oil company, more than four years to develop. It has been in use for a number of years, producing results that are sensibly superior than manual solutions. This success attests to the quality of the solutions produced by the ORCA algorithm.

All tests were run on a platform equipped with a Sun SPARC Ultra 60 processor, running a Solaris 9 operating system at 450 MHz and with 1024 MB of RAM. Both the GRASPW and the ORCA implementations were allowed to run for 1800 seconds on each instance.

3.1 Typical Instances

Twenty two real instances provided by Petrobras were used in our tests. Table 1 summarizes the dataset. Columns with the same numerical data, like columns 8 and 9, refer to distinct instances that differ in the number of other constraints not shown in the table, like C9. The first part of that table displays the instances were no C7 constraints were found. In order to reduce the amount of time spent in testing, in some experiments we used only 7 of these instances, eliminating instances that differed only by a few constraints. The lower part of Table 1 shows the ten instances were C7 constraints were present. It is worth mentioning that, as GRASP makes use of randomization, each instance was tested five times, and the results being reported always reflect the average of the tests.

The time horizon ranges from a thousand to three thousand days. Note that, after every well has been scheduled, the oil yield is the same between all solutions until the time horizon is reached. Thus, any gains in production, when comparing two solutions, happen before all wells are scheduled. If we considered the full horizon to compute yields, these gains, in percent, would be smaller. Therefore, for each instance we use as the horizon of production the end date of the last activity of the well scheduled the latest, among all solutions of the solvers.

3.2 Setting GRASPW Parameters

In Section 2 we presented the idea of a *dynamic sized RCL*. There are at least two ways we could exploit this idea: we may decrease monotonically the number of candidates using a greedier heuristic; or we may increase monotonically the number of candidates in order to drive the search away from a local optimum into new regions of the search tree. In the first case, the initial RCL size was set to $\max(13, w)$, w being the number of wells, and was decreased by one every 300 seconds without improvement. In the second case, we started with $\max(5, w)$ for the initial RCL size and increased it by one every 300 seconds without improvement. The first approach did not yield good results when applied to the WDRDP, generating the same or worse solutions than those found by GRASPW with a static sized RCL. However, the second approach proved promising. Figure 11.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 11.b after 50 thousand iterations, when another real instance is tested. Amongst twelve scenarios tested, four had better solutions with the dynamic sized RCL, summing up an increase of around 261 thousand barrels of oil. In the other eight scenarios, the same solutions were found. The size of the RCL changed around four times in a typical run.

We also considered two options for searching the neighborhood and selecting a new neighbor: the *first improvement rule* (FIR) and the *best improvement rule* (BIR). Tests were executed and FIR proved to be the better. On the other hand, the BIR heuristics found solutions whose production was equal to a predefined target value with the least number of iterations (see Figure 12.a). On 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 12.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. That is because, on the average, a FIR iteration was almost seven times faster than a BIR iteration. Note that, to users, running time is deemed important. Note also that the FIR solver built solutions with a slightly higher oil production, as can be seen in Figures 12 and 13. Summing up all instances, the gain was almost 80 thousand barrels of oil. We concluded that the FIR strategy was the one that better suited this problem.

Another technique tested was to bias the selection towards some particular candidates, those with the highest oil yield. Five probability distributions, besides the uniform distribution, were considered, as indicated in Section 2.2. Comparative experiments showed that the exponential and quadratic bias functions generated much worse solutions, with production of more than 2 millions barrels of oil smaller than the other approaches, summing up over all instances. Among the other bias functions, the one based on the square root proved to be the best one for this problem, as can be seen in Figures 14.a and 15.a. For a better view we compared the square root with the others bias functions. See Figures 14.(b,c,d) and 15.(b,c,d).

3.3 The GRASPW Implementations

In Section 2.2, we presented some advanced techniques that can be included in the basic GRASP procedure. Many combinations of these were tested, but not all possible combinations. With 6 distinct techniques (five advanced plus the pivoting rule), all possible combinations would number 64 distinct solvers. It was simply too time consuming to execute broad tests with each and every one of these solvers. We choose the best combinations based on faster tests, selecting a pool of 27 solvers. To simplify the visualization and data analysis, we show only 14 solvers, besides the ORCA solver. See Table 2.

As there are many solvers and in many tests there were ties between them, we elaborated five criteria to determine which would be the best solver:

1. Sum of the oil production of the best solutions for all instances.

2. Sum of the oil production of the average solutions for all instances.

3. Average of the rank obtained sorting decreasingly the solver solutions of each instance using the *Ranking1 Rule* (R1), explained below. The sorting was according to the average oil production and average time of execution to find the best solution. We ranked all solvers.

4. Average of the rank obtained sorting decreasingly the solver solutions of each instance using the *Ranking2 Rule* (R2), explained below. The sorting was according to the average oil production and average time of execution to find the best solution. We ranked all solvers.

5. Number of instances for which the solver obtained the best solution known to that instance.

Ranking1 Rule is a rank where, if s solvers are drawn in rank r, the next solvers would come in rank $r + 1$. *Ranking2 Rule*, in the same example, would put the next solvers in rank $r + s$.

Table 3 presents the values of the criteria achieved by the 15 solvers created for the WDRDP. Values in boldface are the best results among the solvers for each criteria. For the ease of visualization, criteria 1 and 2 have been normalized from 0.0%, the worse result in each column, to 100%, the best result in each column. In terms of absolute values, solver G13 reached a production of 354,932,391 barrels, according to criteria 1. Solver G14 reached 346,814,967 barrels, according to criteria 2.

Note that G14 stands out as being the best solver according to three criteria, followed closely by G7 which was the best in two criteria. However we can not say yet that G14 is the best overall solver, because if it behaves poorly in the other two criteria another solver could prove to be more appropriate for the WDRDP. Given these values, we elaborated Table 4, where we show the rank of the solvers in each criterion. Note that there are significant variations in the rank of the solvers when using different criteria.

In order to make a decision about which would be the best solver, we again ranked the solvers according to the average rank they obtained in each criterion. We made this rank using both R1 and R2 type rules, as can be seen in Table 5. Using this table, we obtained the rank of the best solvers, presented in Table 6. We conclude that G14 is actually the best GRASP solver that we developed for the WDRDP.

This solver makes use of many advanced techniques for GRASP heuristics, like POP, bias function based on the square root, intensification, path-relinking and SLS (see Section 2.2). Our second best solver for the problem was G7, which makes use only of the POP technique. As can be seen in Table 3, and by the average rank in Table 5, G7 is almost as good as G14. Hence, the other advanced techniques used in solver G14, besides POP, gave rise only to marginal gains. POP, on the other hand, improves drastically the performance of a solver for this problem. This can be seen comparing G1 (basic GRASP) to G7 (basic GRASP with POP) in the same tables mentioned above, or comparing the group of solvers G1 until G6 (solvers without POP) to the group of G7 and the other solvers, that is, the group of solvers that implemented POP. When POP is applied, a great leap in quality can be seen. Note, for example, that G1 is just the eleventh best solver, while G7 is the second best.

Note also that G7 is better than G10, G11, G12 and G13, solvers which combine POP and other advanced techniques. This happened because these advanced techniques required longer processing times, and the gains were not sufficient to justify their use. Solver G7 could only be overcome when all the advanced techniques were combined in G14, and even so by a small margin. With the above facts in mind, we conclude that POP is the best advanced technique that we applied to solve the WDRDP.

4 Conclusions

Scheduling activities efficiently is of paramount importance to the industry, in general. Petrobras, a leading company in deep water oil exploration, presented us the WDRDP, a scheduling problem related to oil well development. Here we contrast two approaches to the WDRDP: the constraint programming tool ORCA and a GRASP implementation, dubbed GRASPW.

As can be seen in Section 3.3, the G14 solver was considered the best GRASPW implementation. Note that the constraint programming solver, ORCA, was much worse than all the GRASPW solvers, even when compared to the standard GRASP procedure, G1. Compared to ORCA, the G14 solver generated a net surplus of more than 16 million barrels of oil, summing up over the average solutions of all instances. This increase of 4,5% in oil production would generate gains of almost US$ 1.14 billion, quoting each barrel at US$ 71.25, when using the G14 solver, instead of the ORCA solver. Besides, even generating better solutions, G14 takes, to find its best solution, 95% of the time ORCA takes to find its best solution. And to find better solutions than the ORCA's best one, G14 takes only 45% of the time taken by ORCA. Among the 17 instances tested, G14 generates, within the first second of execution, better solutions than ORCA's best solution in 14 instances. Two examples of that can be seen in Figure 16. In Figure 17, we show two instances where G14 does not generate, within the first second of execution, solutions better that the ORCA best solution.

However, there is an instance where ORCA was competitive compared to G14, as can be seen in Figure 18. Here, ORCA's best solution was found in 5 seconds and G14 only generated a better solution than that after 1134 seconds. Nevertheless, after 1522 seconds, G14 generated a solution with an oil production almost 63 thousand barrels superior to ORCA's best solution.

The GRASP implementations were built and then tested over several real instances provided by Petrobras. In fact, Section 3.3 could be made much longer, by trying some of the many other possible combinations over the basic GRASP heuristics that were not tried here, such as GRASP+POP+Int and GRASP+POP+P-R, for example. We leave these possibilities as extensions of this work. The combinations studied, however, already support the conclusion that the GRASPW heuristics greatly outperforms the original ORCA approach.

References

Aiex, R. M., Resende, M. G. C. & Toraldo, G. (2005), 'GRASP with path-relinking for the three-index assignment problem', *INFORMS J. on Computing* **17**, 224–247.

Bard, J., Feo, T. & Holland, S. (1996), 'A GRASP for scheduling printed wiring board assembly', *IIE Transactions* **28**, 155–165.

Bard, J. F. & Feo, T. A. (1989), 'Operations sequencing in discrete parts manufacturing', *Management Science* **35**, 249–255.

Binato, S., Hery, W. J., Loewenstern, D. & Resende, M. G. C. (2002), A greedy randomized adaptive search procedure for job shop scheduling, *in* P. Hansen & C. C. Ribeiro, eds, 'Essays and Surveys on Metaheuristics', Kluwer Academic Publishers, pp. 59–79.

Bresina, J. L. (1996), Heuristic-biased stochastic sampling, *in* 'Proceedings of the Thirteenth National Conference on Artificial Intelligence', pp. 271–278.

Canuto, S. A., Resende, M. G. C. & Ribeiro, C. C. (2001), 'Local search with perturbations for the prize-collecting steiner tree problem in graphs', *Networks* **38**, 50–58.

do Nascimento, J. M. (2002), Hybrid computational tools for the optimization of the production of petroleum in deep waters, Master's thesis, Institute of Computing, University of Campinas.

Feo, T. A. & Bard, J. F. (1989), 'Flight scheduling and maintenance base planning', *Management Science* **35**, 1415–1432.

Feo, T. A., Bard, J. & Holland, S. (1995), 'Facility-wide planning and scheduling of printed wiring board assembly', *Operations Research* **43**, 219–230.

Feo, T. A. & Resende, M. G. C. (1995), 'Greedy randomized adaptive search procedures', *Journal of Global Optimization* **6**, 109–133.

Festa, P. & Resende, M. (2002), 'GRASP: An annotated bibliography', *Essays And Surveys In Metaheuristics - Kluwer Academic Publishers* .

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

Glover, F., Laguna, M. & Mart, R. (2000), Fundamentals of scatter search and path relinking, *in* M. Laguna & J. L. Gonzles-Velarde, eds, 'Control and Cybernetics', Vol. 39, Kluwer, pp. 653–684.

Glover, F. & Laguna, M. (1997), *Tabu Search*, Kluwer Academic Publishers, Norwell, Massachusetts.

Glover, F. (1996), Tabu search and adaptive memory programming — advances, applications and challenges, *in* R. S. Barr, R. V. Helgason & J. L. Kennington, eds, 'Interfaces in Computer Science and Operations Research', Kluwer, pp. 1–75.

Glover, F. (2000), Multi-start and strategic oscillation methods — principles to exploit adaptive memory, *in* M. Laguna & J. L. Gonzles-Velarde, eds, 'Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research', Kluwer, pp. 1–24.

Harvey, W. D. & Ginsberg, M. L. (1995), 'Limited discrepancy search', *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* **1**, 607–613.

ILOG (1999), *ILOG Solver 4.4 Reference Manual*, ILOG.

Marriott, K. & Stuckey, P. J. (1998), *Programming with Constraints: An introduction*, MIT Press, Cambridge, Massachusetts.

Pereira, R. A., Moura, A. V. & de Souza, C. C. (2005), Comparative experiments with GRASP and constraint programming for the oil well drilling problem, *in* S. E. Nikoletseas, ed., 'Experimental and Efficient Algorithms - WEA 2005, LNCS 3503', Springer, pp. 328–340.

Petrobras (1999), 'ORCA'. A constraint programming solution to the WDRDP developed in-house by Petrobras. No known publications in the open literature.

Resende, M. G. C. & Ribeiro, C. C. (2002), Greedy randomized adaptive search procedure, *in* F. Grover & G. Kochenberger, eds, 'Handbook of Metaheuristics', Kluwer, pp. 219–249.

Resende, M. G. C. & Ribeiro, C. C. (2003), 'GRASP with path-relinking for private virtual circuit routing', *Networks* **41**, 104–114.

Roy, B. & Sussmann, B. (1964), Les problmes d'ordonnancement avec contraintes disjonctives, *in* 'Note DS No 9 bis', SEMA, Paris.

Yannakakis, M. (1997), Computational complexity, *in* E. H. L. Aarts & J. K. Lenstra, eds, 'Local Search in Combinatorial Optimization', John Wiley & Sons, Chichester, pp. 19–55.

5 Tables

Instance	1	2	3	4	5	6	7	8	9	10	11	12
# wells	29	22	29	29	17	22	22	29	29	22	29	22
# activities	98	107	98	98	111	107	128	98	98	107	98	107
# boats	1	1	2	1	1	2	1	1	1	2	1	2
# derricks	3	2	3	3	2	2	3	3	3	2	3	2
# C7	0	0	0	0	0	0	0	0	0	0	0	0
Avg. Outf. (barrels)	2321	338	3026	3026	2320	3026	3026	2671	3026	3026	2671	3026
Avg. Act. Dur. (days)	27	21	19	21	27	21	27	27	21	27	27	21

Instance	13	14	15	16	17	18	19	20	21	22
# wells	22	22	22	22	22	22	22	22	29	65
# activities	107	107	107	107	107	107	107	107	98	338
# boats	2	2	2	1	2	2	2	2	1	1
# derricks	2	2	2	2	2	2	2	2	3	2
# C7	1	1	1	1	1	1	1	1	2	2
Avg. Outf.(barrels)	3026	3026	3026	3026	3026	3026	3026	3026	2321	4028
Avg. Act. Dur. (days)	21	21	21	21	21	21	21	21	27	12

Table 1: Tested Instances.

Techniques	C.P.	GRASP	POP	Bias	Int	P-R	SLS	BIR
ORCA	●							
G1		●						
G2		●		●				
G3		●		●	●			
G4		●				●		
G5		●					●	
G6		●						●
G7		●	●					
G8		●	●				●	
G9		●	●					●
G10		●	●	●				
G11		●	●	●			●	
G12		●	●	●			●	●
G13		●	●	●	●		●	
G14		●	●	●	●	●	●	

C.P.	Constraint Programming
GRASP	Greedy Randomized Adaptive Search Procedure
POP	Proximate Optimality Principle
Bias	Bias Function (Square Root)
Int	Intensification
P-R	Path-Relinking
SLS	Selective Local Search
BIR	BIR

Table 2: Solvers

Criterion	1	2	3	4	5
ORCA	0,00	0,00	9,588235	23,47059	2
G1	97,89	96,90	7,682353	10,47059	7
G2	97,86	96,46	8,035294	11,11765	6
G3	98,31	96,85	7,470588	10,23529	6
G4	97,69	96,45	8	11,41176	6
G5	97,89	97,01	6,823529	9,176471	7
G6	97,91	97,23	4,623529	8,235294	7
G7	99,61	99,95	**2,941176**	**4,352941**	8
G8	99,61	99,55	4,352941	5,764706	8
G9	99,61	99,92	3,470588	5,294118	8
G10	99,99	99,79	4,411765	6,588235	7
G11	99,99	99,67	4,294118	6,411765	**9**
G12	99,99	99,66	4,588235	6,941176	7
G13	**100,00**	99,69	4,058824	6	7
G14	99,81	**100,00**	**2,941176**	4,764706	**9**

Table 3: Values obtained by the Solvers in the Comparative Criteria

Criterion	1	2	3	4	5
	G13	G14	G7-G14	G7	G11-G14
	G10-G11-G12	G7	G9	G14	G7-G8-G9
	G14	G9	G13	G9	G1-G5-G6-G10-G12-G13
	G7-G8-G9	G10	G11	G8	G2-G3-G4
	G3	G13	G8	G13	ORCA
	G6	G11	G10	G11	
	G1-G5	G12	G12	G10	
	G2	G8	G6	G12	
	G4	G6	G5	G6	
	ORCA	G5	G3	G5	
		G1	G1	G3	
		G3	G4	G1	
		G2	G2	G2	
		G4	ORCA	G4	
		ORCA		ORCA	

Table 4: Rank of the Solvers

Analysis of Criteria	R1	R2	Average R1-R2
ORCA	11,8	15	13,4
G1	8,8	10,4	9,6
G2	10,2	13	11,6
G3	8,4	11	9,7
G4	10,6	13,4	12
G5	7,8	9,4	8,6
G6	7	8,6	7,8
G7	2	2,6	2,3
G8	4,6	5,4	5
G9	2,8	3,6	3,2
G10	4,4	5,2	4,8
G11	3,8	4	3,9
G12	5,4	6,2	5,8
G13	3,4	4,2	3,8
G14	**1,6**	**2**	**1,8**

Table 5: Analysis of the Rank of the Solvers

Solver
G14
G7
G9
G13
G11
G10
G8
G12
G6
G5
G1
G3
G2
G4
ORCA

Table 6: Rank of the Best Solvers

6 Figures

```
1: procedure $GRASP(ListSize, MaxIter, Seed)$
2: for $k = 1$ to $MaxIter$ do
3:     $Solution \leftarrow Construct\_Solution(ListSize, Seed)$;
4:     $Solution \leftarrow Local\_Search(Solution)$;
5:     $Update\_Solution(Solution, Best\_Solution\_Found)$;
6: end for
7: return $Best\_Solution\_Found$;
8: end $GRASP$
```

Figure 1: The GRASP Meta-heuristics.

```
1: procedure $Construct\_Solution(ListSize, Seed)$
2: $Solution \leftarrow 0$;
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:     $Solution \leftarrow Solution \cup \{s\}$;
8:     $Reevaluate\ the\ incremental\ costs$;
9: end while
10: $return\ Solution$;
11: $end\ Construct\_Solution$
```

Figure 2: The Construction Phase of GRASP.

```
procedure $searchForSolutions()$
while $Time\ limit\ is\ not\ reached$ do
    if $Time\ limit\ with\ no\ improvements\ is\ reached$ then
        $RCLSize \leftarrow RCLSize + 1$;
    end if
    $Solution \leftarrow greedyRandomSolution()$;
    if $Production\ of\ Solution \geq\ least\ production\ of\ the\ elite\ solutions$ OR $Solution\ differs\ from\ elite\ solutions$ then
        $doTwoExchangeLocalSearch(Solution)$;
    end if
    $doPathRelinking(Solution)$;
    if $Solution\ is\ an\ elite\ solution$ then
        $Insert\ Solution\ into\ the\ elite\ set$;
    end if
    if $Production\ of\ Solution\ is\ the\ highest\ so\ far$ then
        $BestSolution \leftarrow Solution$;
    end if
end while
$return\ BestSolution$;
$end\ searchForSolutions$
```

Figure 3: The new GRASP solver implementation.

```
1:  procedure $greedyRandomSolution()$
2:  $//Schedule\ first\ production\ wells\ and\ injection\ wells\ that\ precede\ production\ wells;$
3:  while $Not\ all\ production\ wells\ were\ scheduled\ in\ Solution$ do
4:      $doPOP(Solution);$
5:      for $Well$ in $List\ of\ wells\ availables$ do
6:          if $A\ start\ date\ of\ any\ activity\ of\ Well\ has\ very\ small\ domain$ then
7:              $schedule(Solution, Well);$
8:          end if
9:      end for
10:     $WellsToSchedule \leftarrow Wells\ availables\ that\ produces\ or\ with\ producer\ successors;$
11:     $Sort\ WellsToSchedule\ decreasingly\ by\ estimated\ production;$
12:     $RCL \leftarrow First\ 'RCLSize'\ wells\ from\ WellsToSchedule;$
13:     $WellChoosen \leftarrow Choose\ well\ from\ RCL\ randomly\ with\ bias;$
14:     $schedule(Solution, WellChoosen);$
15:     if $Solution\ does\ not\ satisfy\ constraints$ then
16:         $return\ empty\ solution;$
17:     end if
18: end while
19: $Schedule\ the\ remaining\ wells,\ the\ injection\ wells;$
20: if $Solution\ does\ not\ satisfy\ constraints$ then
21:     $return\ empty\ solution;$
22: else
23:     $return\ Solution;$
24: end if
25: $end\ greedyRandomSolution$
```

Figure 4: The new Greedy Randomized procedure.

```
1: procedure $doPOP(Solution)$
2: if $\%$ $of\ activities\ scheduled\ in\ Solution\ is\ a\ multiple\ of\ the\ POP\ frequency$ then
3:     $doTwoExchangeLocalSearch(Solution)$;
4: end if
5: end doPOP
```

Figure 5: The POP procedure.

```
procedure $doPathRelinking(Solution)$
$EliteSolution \leftarrow Get\ a\ random\ solution\ from\ the\ elite\ set$;
$Current \leftarrow Best\ solution\ between\ Solution\ and\ EliteSolution$;
$Guide \leftarrow Worst\ solution\ between\ Solution\ and\ EliteSolution$;
for $Resource$ in $List\ of\ resources$ do
    for $Well$ in $List\ of\ wells\ consecutively\ scheduled\ in\ Resource\ in\ Guide$ do
        $//Puts\ Well\ in\ Current\ in\ the\ same\ position\ it\ appears\ in\ Guide$
        $Position \leftarrow Position\ where\ Well\ is\ scheduled\ in\ Resource\ in\ Guide$;
        $Wegll2 \leftarrow Get\ well\ scheduled\ in\ position\ Position\ in\ Resource\ in\ Current$;
        $Swap\ wells\ Well\ and\ Well2\ in\ Current$;
        if $Production\ of\ Current\ is\ the\ highest\ so\ far$ then
            $BestSolution \leftarrow Current$;
        end if
    end for
end for
end doPathRelinking
```

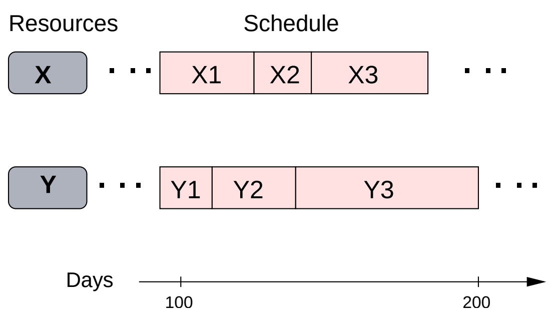
Figure 6: The Path-Relinking procedure.

```
procedure $doTwoExchangeLocalSearch(Solution)$
for $Well$ in $List\ of\ wells\ scheduled\ in\ Solution$ do
    for $Well2$ in $List\ of\ wells\ scheduled\ in\ Solution\ continuing\ from\ Well$ do
        $Swap\ wells\ Well\ and\ Well2\ in\ Solution$;
        if $Production\ of\ Solution\ is\ the\ highest\ so\ far$ then
            $BestSolution \leftarrow Solution$;
        end if
    end for
end for
end $doTwoExchangeLocalSearch$
```

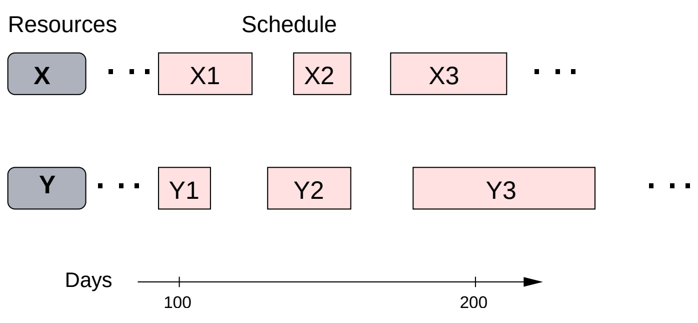
Figure 7: The Two-Exchange Local Search procedure.

```
procedure $schedule(Solution, Well)$
for $Activity$ in $List\ of\ activities\ availables\ of\ Well\ ordered\ by\ constraints$ do
    $Resource \leftarrow Resource\ able\ to\ do\ Activity\ the\ earliest,\ preferring\ last\ used\ in\ Well$;
    $scheduleInResource(Solution, Activity, Resource)$;
    if $Well\ produces$ then
        $Adds\ the\ production\ of\ Well\ to\ the\ production\ of\ Solution$;
    end if
end for
$end\ schedule$
```

Figure 8: The Schedule procedure.

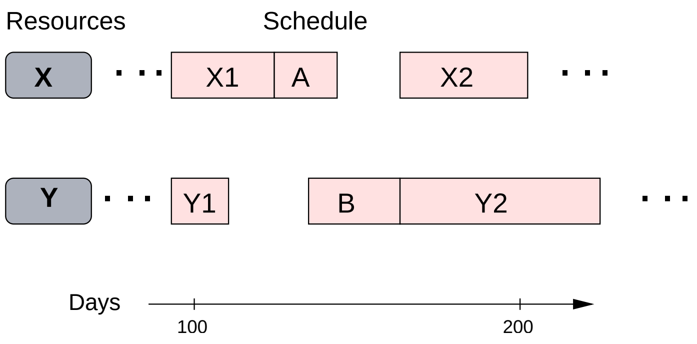


(a) No Displacement

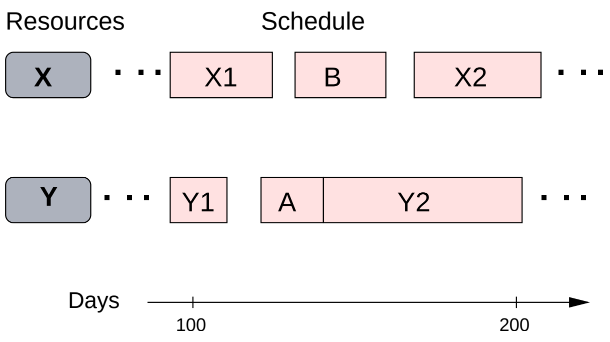


(b) With Displacement

Figure 9: Schedules without and with resource displacement.

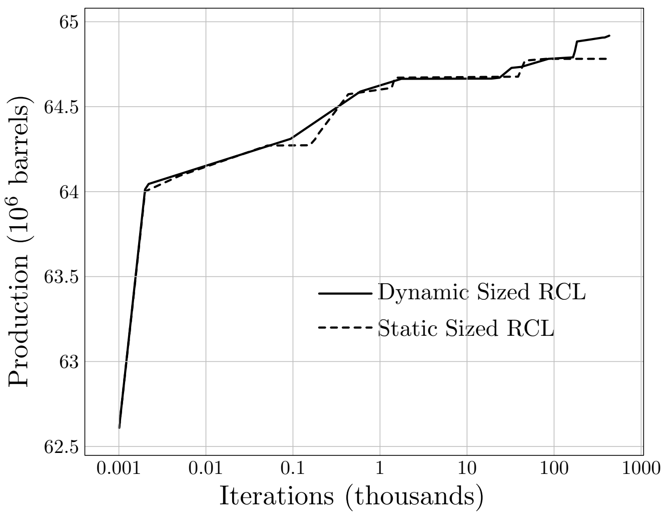


(a) Before

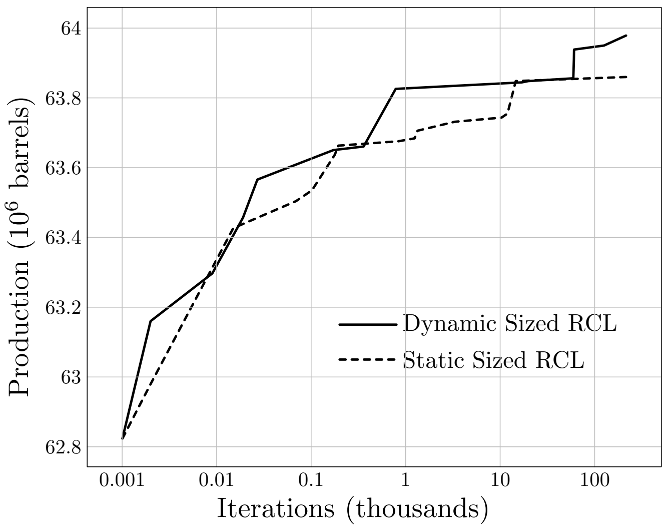


(b) After

Figure 10: 2-exchange swap.

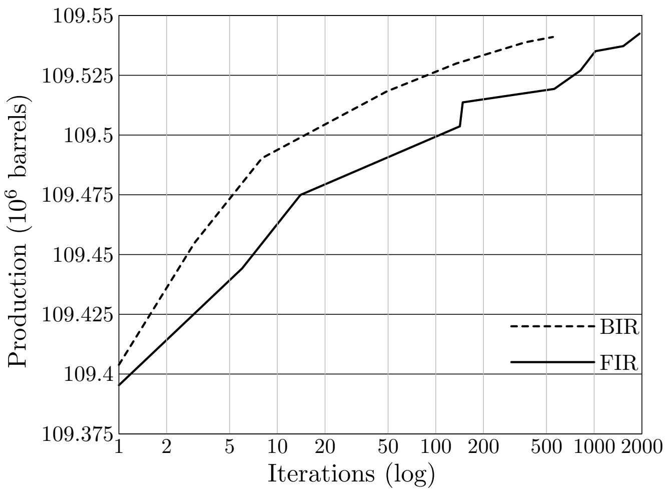


(a) Example 1 – Instance 4

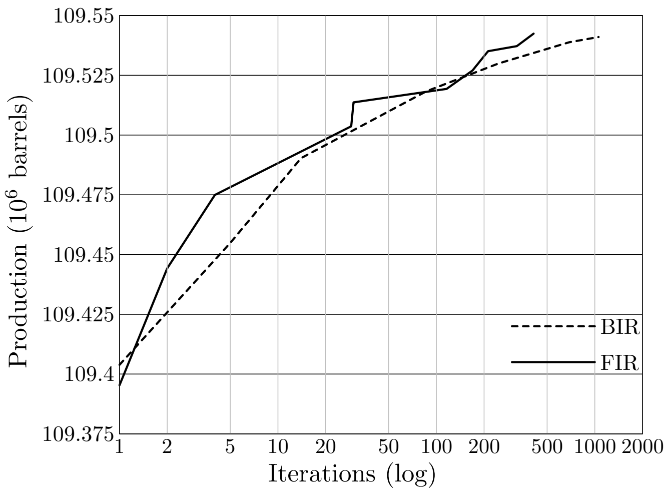


(b) Example 2 – Instance 6

Figure 11: Static Sized RCL Solver x Dynamic Sized RCL Solver.

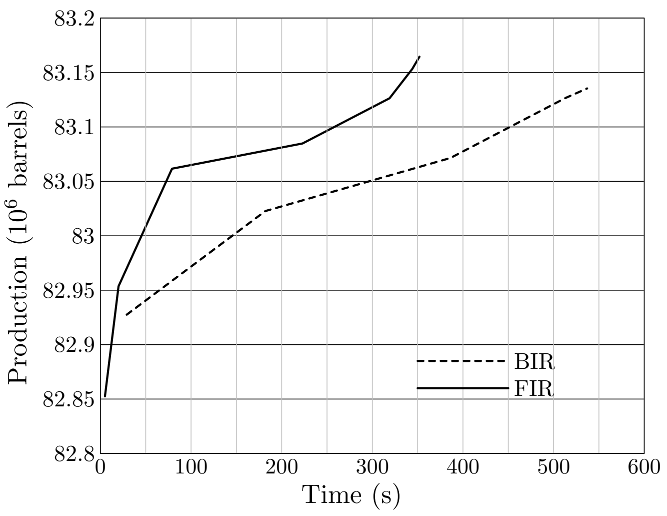


(a) Using instance 21

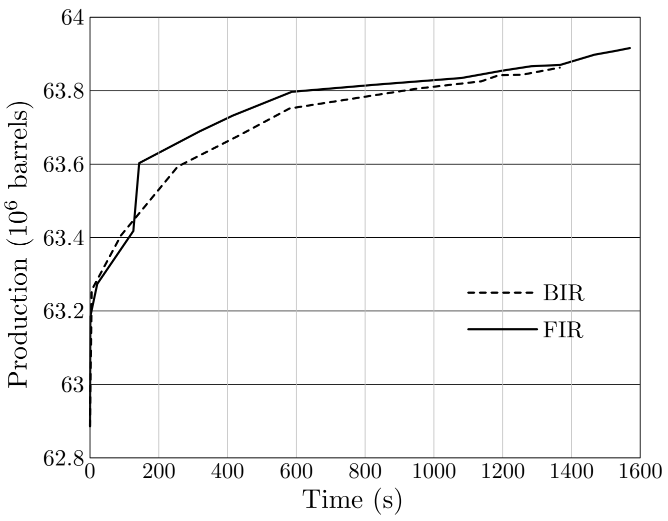


(b) Using instance 21

Figure 12: BIR x FIR: Example 1

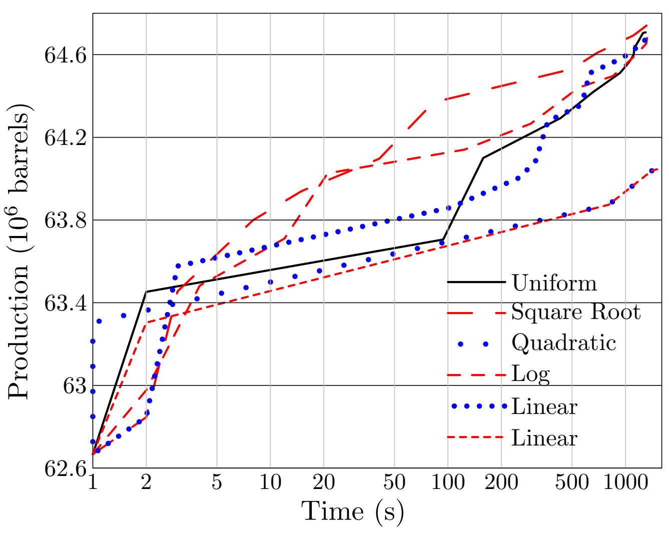


(a) Example 2 – Instance 22

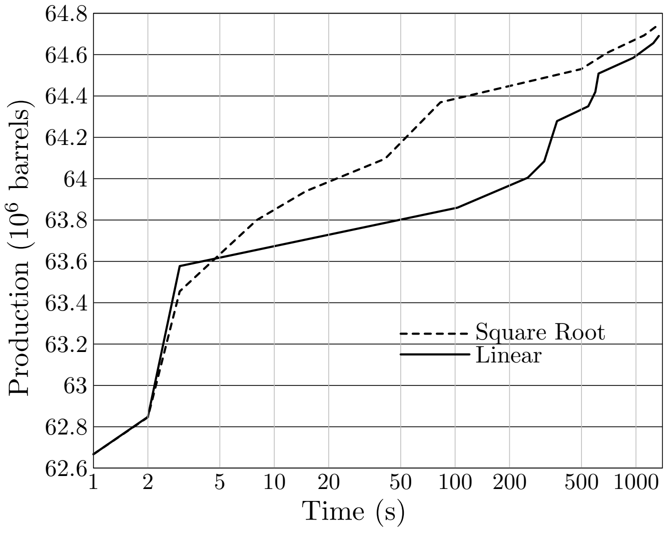


(b) Example 3 – Instance 4

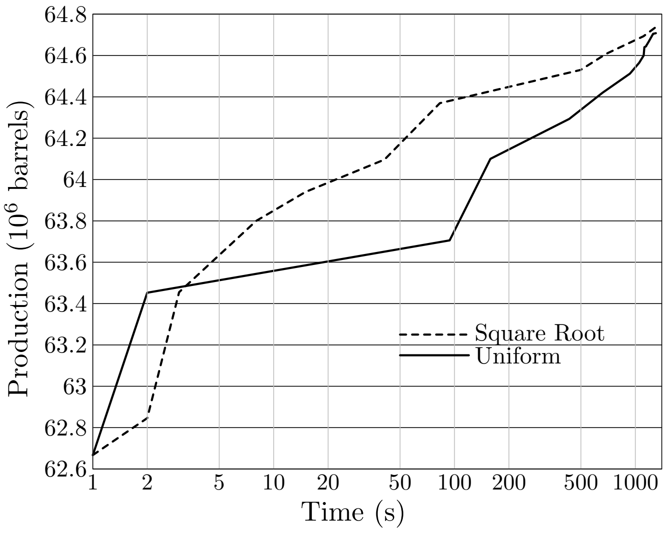
Figure 13: BIR x FIR.



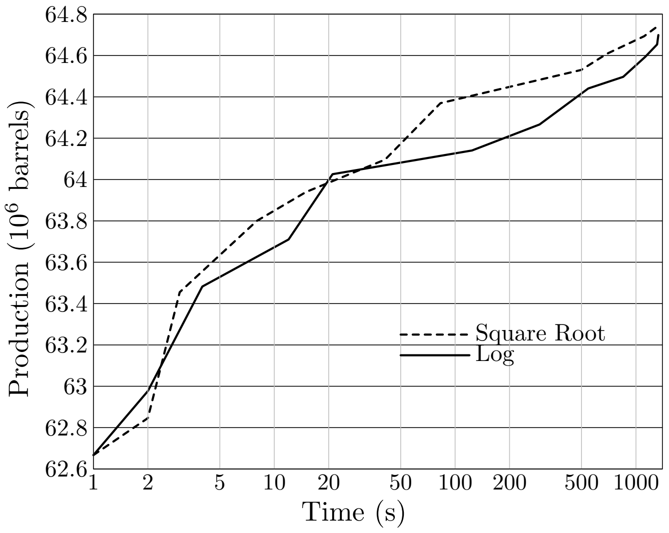
(a) Example 1– Instance 3



(b) Square root X Linear– Instance 3

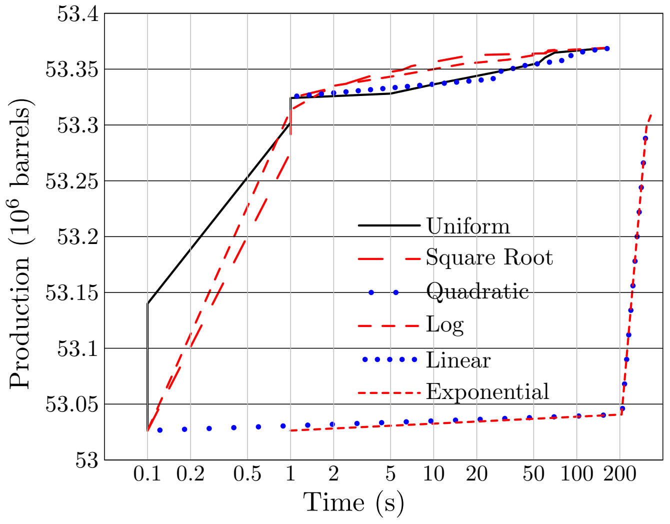


(c) Square root X Uniform – Instance 3

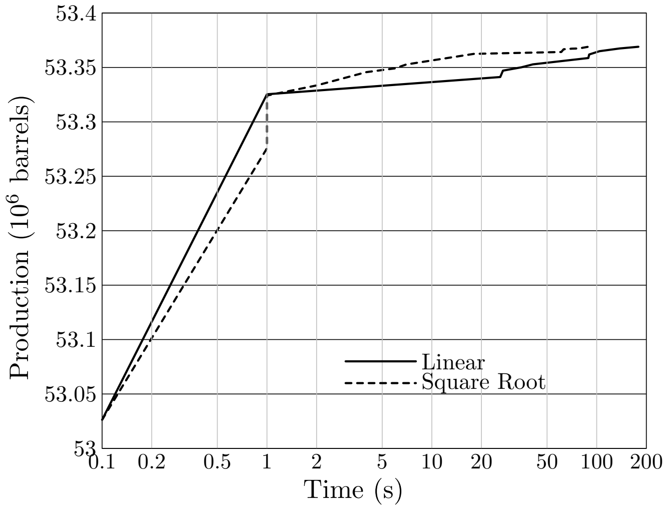


(d) Square root X Log – Instance 3

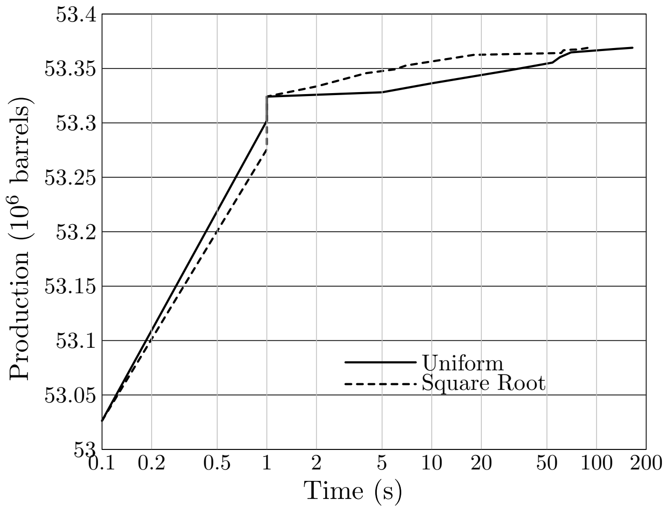
Figure 14: Bias Functions, I.



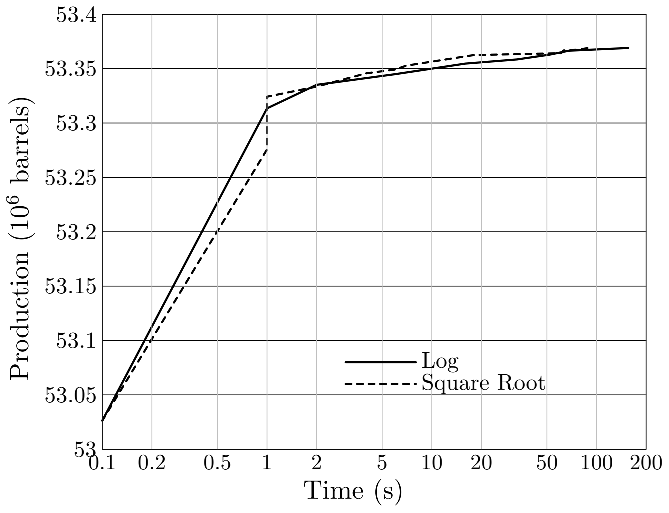
(a) Example 2 – Instance 8



(b) Square root X Linear – Instance 8

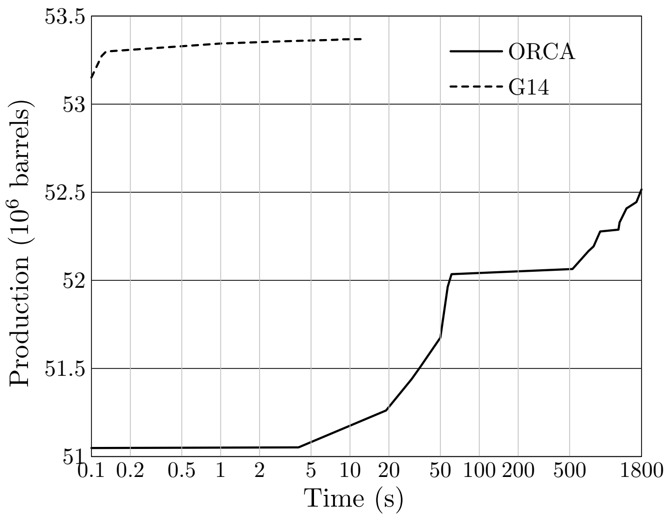


(c) Square root X Uniform – Instance 8

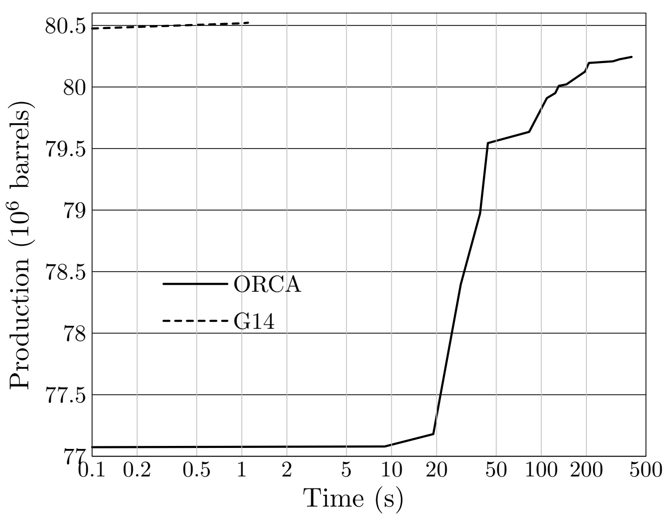


(d) Square root X Log – Instance 8

Figure 15: Bias Functions, II

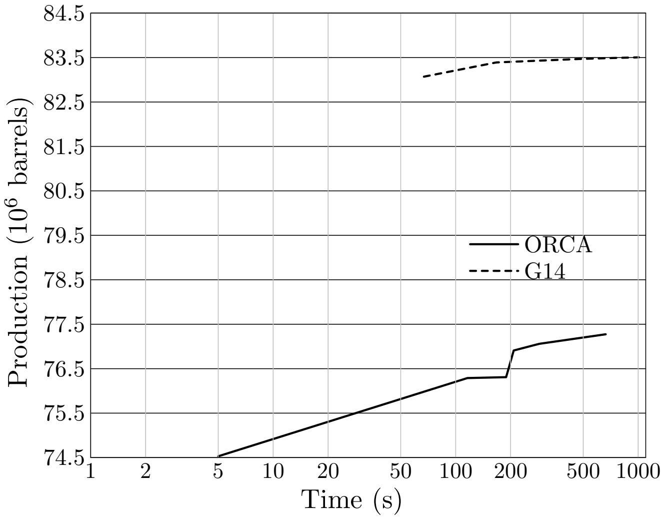


(a) Example 1 – Instance 8

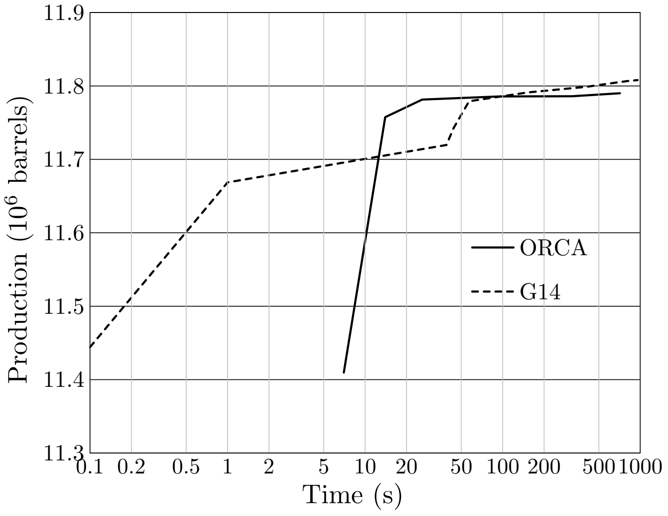


(b) Example 2 – Instance 11

Figure 16: ORCA × G14, I



(a) Example 3 – Instance 22



(b) Example 4 – Instance 15

Figure 17: ORCA × G14, II

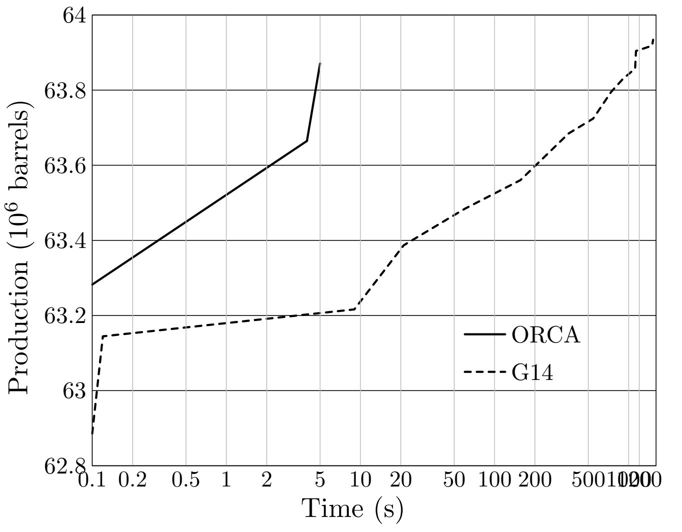


Figure 18: ORCA×G14 – Instance 4