A GRASP FOR JOB SHOP SCHEDULING

S. BINATO, W.J. HERY, D.M. LOEWENSTERN, AND M.G.C. RESENDE

Abstract. In the job shop scheduling problem (JSP), a finite set of jobs is processed on a finite set of machines. Each job is characterized by a fixed order of operations, each of which is to be processed on a specific machine for a specified duration. Each machine can process at most one job at a time and once a job initiates processing on a given machine it must complete processing uninterrupted. A schedule is an assignment of operations to time slots on the machines. The objective of the JSP is to find a schedule that minimizes the maximum completion time, or makespan, of the jobs. In this paper, we describe a greedy randomized adaptive search procedure (GRASP) for the JSP. A GRASP is a metaheuristic for combinatorial optimization. Although GRASP is a general procedure, its basic concepts are customized for the problem being solved. We describe in detail our implementation of GRASP for job shop scheduling. Further, we incorporate to the conventional GRASP two new concepts: an intensification strategy and POP (Proximate Optimality Principle) in the construction phase. These two concepts were first proposed by Fleurent & Glover (1999) in the context of the quadratic assignment problem. Computational experience on a large set of standard test problems indicates that GRASP is a competitive algorithm for finding approximate solutions of the job shop scheduling problem.

1. Introduction

In the job shop scheduling problem (JSP), a finite set of jobs is processed on a finite set of machines. Each job is characterized by a fixed order of operations, each of which is to be processed on a specific machine for a specified duration. Each machine can process at most one job at a time and once a job initiates processing on a given machine it must complete processing on that machine uninterrupted. A schedule is an assignment of operations to time slots on the machines. The makespan is the maximum completion time of the jobs. The objective of the JSP is to find a schedule that minimizes the makespan.

Formally, the JSP can be stated as follows. Given a set $\mathcal{M}$ of machines (denote the size of $\mathcal{M}$ by $|\mathcal{M}|$) and a set $\mathcal{J}$ of jobs (denote the size of $\mathcal{J}$ by $|\mathcal{J}|$), let $\sigma_1^j \prec \sigma_2^j \prec \cdots \prec \sigma_{|\mathcal{M}|}^j$ be the ordered set of $|\mathcal{M}|$ operations of job j, where $\sigma_k^j \prec \sigma_{k+1}^j$ indicates that operation σ_{k+1}^j can only start processing after the completion of operation σ_k^j. Let $\mathcal{O}$ be the set of operations. Each operation σ_k^j is defined by two parameters: $\mathcal{M}_k^j$ is the machine on which σ_k^j is processed and $p_k^j = p(\sigma_k^j)$ is the processing time of operation σ_k^j. Defining $t(\sigma_k^j)$ to be the starting time of the k-th operation $\sigma_k^j \in \mathcal{O}$, a disjunctive programming formulation for the JSP is as follows:

Date: March 2000.

Key words and phrases. Combinatorial optimization, job shop scheduling, local search, GRASP, intensification, proximal optimality principle, probabilistic algorithm.

AT&T Labs Research Technical Report: 00.6.1.

$$\min C_{\max}$$

subject to:

$$C_{\max} \geq t(\sigma_k^j) + p(\sigma_k^j), \text{ for all } \sigma_k^j \in \mathcal{O},$$

$$t(\sigma_k^j) \geq t(\sigma_l^j) + p(\sigma_l^j), \text{ for all } \sigma_l^j \prec \sigma_k^j, \tag{1a}$$

$$t(\sigma_k^j) \geq t(\sigma_l^i) + p(\sigma_l^i) \vee \tag{1b}$$

$$t(\sigma_l^i) \geq t(\sigma_k^j) + p(\sigma_k^j), \text{ for all } i, j \in \mathcal{J} \ni \mathcal{M}_{\sigma_l^i} = \mathcal{M}_{\sigma_k^j},$$

$$t(\sigma_k^j) \geq 0, \text{ for all } \sigma_k^j \in \mathcal{O}.$$

C_{max} is the makespan to be minimized.

A feasible solution of the JSP can be built from a permutation of $\mathcal{J}$ on each of the machines in $\mathcal{M}$, observing the precedence constraints, the restriction that a machine can process only one operation at a time, and requiring that once started, processing of an operation must be uninterrupted until its completion. Each set of permutations has a corresponding schedule. Thus, the objective of the JSP is to find a set of permutations with the smallest makespan.

The JSP is NP-hard [20] and has continuously challenged computational researchers. Even instances with three machines and unit processing times, as well as instances with three jobs, are NP-hard. If preemption is allowed, the JSP remains NP-hard. Exact methods [3, 6, 7, 8, 16] have been successful in solving small instances, including the notorious 10×10 instance of Fisher and Thompson [13], proposed in 1963 and only solved twenty years later. Problems of dimension 15×15 are usually considered to be beyond the reach of exact methods. For such problems there is a need for good heuristics. Surveys of heuristic methods for the JSP are given in [22, 26]. These include dispatching rules reviewed in [15], the shifting bottleneck approach [1, 3], local search [26], simulated annealing [27], tabu search [25, 21], and genetic algorithms [9]. A comprehensive survey of job shop scheduling techniques can be found in Jain and Meeran [18]. In this paper we present a greedy randomized adaptive search procedure (GRASP) for the job shop scheduling problem.

The remainder of the paper is organized as follows. In Section 2, we make a brief review of the building blocks of GRASP. Section 3 focuses on a basic GRASP for the job shop scheduling problem, describing a construction mechanism and a local search algorithm. In Sections 4 and 5, an intensification scheme that makes use of memory mechanisms and the use of the proximal optimality principle are incorporated to the basic GRASP. Computational results are reported in Section 6 and concluding remarks are made in Section 7.

2. A brief review of GRASP

In this paper, we apply the concepts of GRASP (greedy randomized adaptive search procedures) to the job shop scheduling problem. GRASP [10, 11] is an iterative process, where each GRASP iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is explored by local search. The best solution over all GRASP iterations is returned as the result.

```
procedure GRASP(ListSize,MaxIter,RandomSeed)
1    do $k = 1, \dots ,$ MaxIter $\rightarrow$
2        ConstructSolution(ListSize,RandomSeed);
3        LocalSearch(BestSolutionFound);
4        UpdateSolution(BestSolutionFound);
5    od;
6    return BestSolutionFound
end GRASP;
```

FIGURE 1. A generic GRASP pseudo-code.

In the construction phase, a feasible solution is built, one element at a time. 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 (myopic) benefit of selecting each element. This list is called the RCL (restricted candidate list). 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 of a GRASP 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 GRASP iteration, while not necessarily jeopardizing the adaptive greedy component.

The solutions generated by a GRASP construction phase are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution from its neighborhood. It terminates when there is no better solution found in the neighborhood with respect to some cost function.

Figure 1 illustrates a generic GRASP implementation in pseudo-code. Input for GRASP includes parameters for setting the candidate list size and the maximum number of GRASP iterations, and the seed for the random number generator. The GRASP iterations are carried out in lines 1–5. Each GRASP iteration consists of the construction phase (line 2), the local search phase (line 3) and, if necessary, the incumbent solution update (line 4).

GRASP has been applied successfully to numerous combinatorial optimization problems, including set covering, quadratic assignment, satisfiability, vehicle routing, location problems, maximum independent set, feedback vertex set, transmission network expansion planning, and graph planarization. For an annotated bibliography of GRASP, see Festa and Resende [12].

3. GRASP for Job Shop Scheduling

In this section, we specialize GRASP for the job shop scheduling problem (JSP). We first describe a basic construction scheme and then show a local search algorithm that uses disjunctive graphs.

3.1. **Construction phase.** The GRASP construction phase builds a feasible solution, one element at a time. In the JSP, we consider a single operation to be the building block of the construction phase. That is, we build a feasible schedule by scheduling individual operations, one at a time, until all operations are scheduled.

Recall that σ_k^j denotes the k-th operation of job j and is defined by the pair $(\mathcal{M}_k^j, p_k^j)$, where $\mathcal{M}_k^j$ is the machine on which operation σ_k^j is performed and p_k^j is the processing time of operation σ_k^j.

While constructing a feasible schedule, not all operations can be selected at a given stage of the construction. An operation σ_k^j can only be scheduled if all prior operations of job j have already been scheduled. Therefore, at each construction phase iteration, at most $|\mathcal{J}|$ operations are candidates to be scheduled. Let this set of candidate operations be denoted by $\mathcal{O}_c$.

More than one greedy algorithm can be proposed for the JSP. One such greedy algorithm selects the operation σ_k^j that results in the smallest increase in the makespan of the already scheduled jobs to schedule next. Let the adaptive greedy function $h(\sigma)$ denote the makespan resulting from the addition of operation σ to the already scheduled operations. The greedy choice is to next schedule operation

$$\underline{\sigma} = \text{argmin}(h(\sigma) \mid \sigma \in \mathcal{O}_c).$$

Defining

$$\overline{\sigma} = \text{argmax}(h(\sigma) \mid \sigma \in \mathcal{O}_c),$$

$\underline{h} = h(\underline{\sigma})$, and $\overline{h} = h(\overline{\sigma})$, the GRASP restricted candidate list (RCL) is defined as

$$\text{RCL} = \{\sigma \in \mathcal{O}_c \mid \underline{h} \leq h(\sigma) \leq \underline{h} + \alpha(\overline{h} - \underline{h})\},$$

where α is parameter such that $0 \leq \alpha \leq 1$.

The next operation to be scheduled is chosen at random from the RCL. In a standard GRASP, the candidates in the RCL are assigned equal probabilities of being chosen. However, any probability distribution can be used to bias the selection towards some particular candidates. Bresina [5] introduced a family of such probability distributions. In Bresina's selection procedure, the candidates are ranked according to the greedy function. Let $r(\sigma)$ denote the rank of element σ and let $\texttt{bias}(r(\sigma))$ be the bias function. The probability $\pi(\sigma)$ of selecting operation σ is

$$\pi(\sigma) = \frac{\texttt{bias}(r(\sigma))}{\sum_{\sigma' \in \text{RCL}} \texttt{bias}(r(\sigma'))}. \tag{2}$$

Bresina introduced several bias functions, which we restrict to elements of the RCL. In random bias, $\texttt{bias}(r) = 1$, for $r \in$ RCL. In linear bias, $\texttt{bias}(r) = 1/r$, for $r \in$ RCL. In log bias, $\texttt{bias}(r) = \log^{-1}(r+1)$, for $r \in$ RCL. In exponential bias, $\texttt{bias}(r) = e^{-r}$, for $r \in$ RCL. In polynomial bias of order n, $\texttt{bias}(r) = r^{-n}$, for $r \in$ RCL. Note that the standard GRASP uses a random bias function.

A typical iteration of the GRASP construction is summarize as follows: a partial schedule (which is initially empty) is on hand, the next operation to be scheduled is selected from the RCL and is added to the partial schedule, resulting in a new partial schedule. Construction ends when the partial schedule is complete, i.e. all operations have been scheduled.

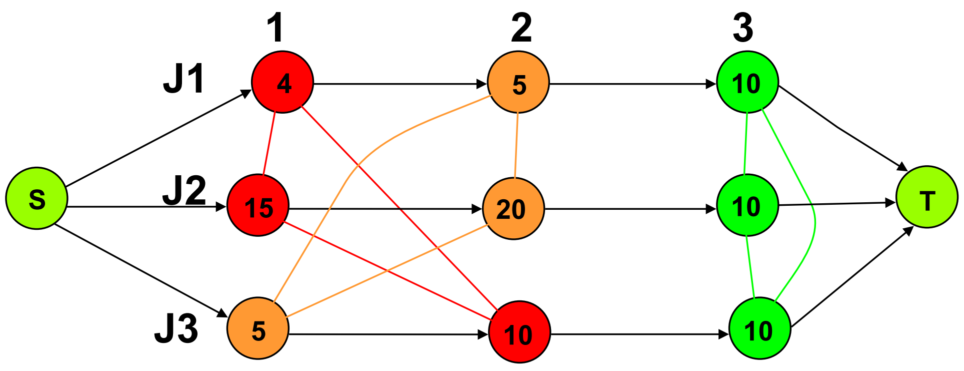

FIGURE 2. Disjunctive graph: representation of schedule.

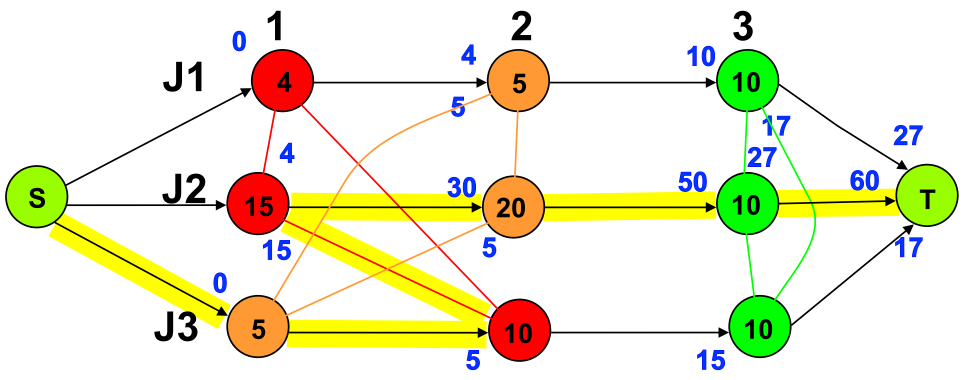

FIGURE 3. Disjunctive graph: critical path with a makespan of 60.

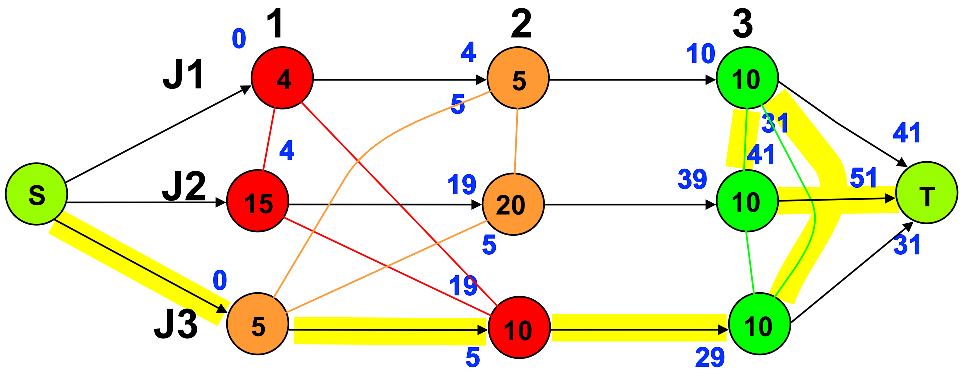

FIGURE 4. Disjunctive graph: recomputing the critical path with a makespan of 51.

3.2. **Local search phase.** Since there is no guarantee that the schedule obtained in the construction phase is locally optimal with respect to the local neighborhood being adopted, local search may improve upon the constructed solution.

The local search algorithm employed in this GRASP for JSP is the two exchange local search based on the disjunctive graph model of Roy and Sussmann [24]. In this model, the disjunctive graph $G = (V, A, E)$ is defined such that

$$V = \{\mathcal{O} \cup \{0, |\mathcal{J}| \cdot |\mathcal{M}| + 1\}\}$$

is the set of nodes, where $\{0\}$ and $\{|\mathcal{J}| \cdot |\mathcal{M}| + 1\}$ are artificial source and sink nodes, respectively,

$$\begin{aligned} A = \{(v,w) \mid v,w \in \mathcal{O}, v \prec w\} \cup & \\ \{(0,w) \mid w \in \mathcal{O}, \nexists v \in \mathcal{O} \ni v \prec w\} \cup & \\ \{(v, |\mathcal{J}| \cdot |\mathcal{M}| + 1) \mid v \in \mathcal{O}, \nexists w \in \mathcal{O} \ni w \prec v\} & \end{aligned}$$

is the set of directed arcs connecting consecutive operations of the same job, and

$$E = \{(v,w) \mid \mathcal{M}_v = \mathcal{M}_w\}$$

is the set of edges that connect operations on the same machine. Vertices in the disjunctive graph model are weighted. Vertices 0 and $|\mathcal{J}| \cdot |\mathcal{M}| + 1$ have weight zero, while the weight of vertex $i \in \{1, \dots, |\mathcal{J}| \cdot |\mathcal{M}|\}$ is the processing time of the operation corresponding to vertex i. Notice that the edges of A and E correspond, respectively, to constraints (1a) and (1b) of the disjunctive programming formulation of the JSP. An example of a disjunctive graph for a 3-job, 3-machine instance is shown in Figure 2.

A feasible schedule consists in any orientation of the edges in E. Given an orientation of E, one can compute the earliest start time of each operation by computing the longest (weighted) path from node 0 to the node corresponding to the operation. Consequently, the makespan of the schedule can be computed by finding the critical (longest) path from node 0 to node $|\mathcal{J}| \cdot |\mathcal{M}| + 1$, as illustrated in Figure 3. Thus, the objective of the JSP is to find an orientation of E such that the longest path in G is minimized.

Taillard [25] describes an $O(|\mathcal{J}| \cdot |\mathcal{M}|)$ algorithm to compute the longest path on G and an $O(|\mathcal{J}| \cdot |\mathcal{M}|)$ procedure to recompute the makespan when two consecutive operations in the critical path (on the same machine) are swapped. We use these procedures in our local search.

The local search procedure begins by identifying the critical path in the disjunctive graph corresponding to the schedule produced in the construction phase. All pairs of consecutive operations sharing the same machine in the critical path are tentatively exchanged. If the exchange improves the makespan, it is accepted. Otherwise, the exchange is undone. Once an exchange is accepted, the critical path may change and a new critical path must be identified. If no pairwise exchange of consecutive operations in the critical path improves the makespan, the current schedule is locally optimal and the local search ends. For example, in Figure 4, the second operation of job 3 was exchanged with the first operation of job 1, resulting in an improvement of the makespan.

4. Intensification

One possible shortcoming of the standard GRASP framework is the independence of the GRASP iterations, i.e. the fact that GRASP does not *learn* from the history of solutions found in previous iterations. This is so because the standard GRASP discards information about any solution encountered that does not improve upon the incumbent. An obvious use of the information obtained from the "good" solutions is to implement a memory-based procedure to influence the construction phase by modifying the probabilities assigned to each RCL element, see eq. (2). Fleurent and Glover [14] introduced one such memory-based scheme. Their approach was illustrated in the context of the quadratic assignment problem (QAP), but is general and can be adapted to multistart methods, such as GRASP, for other combinatorial optimization problems. In this section, we present an intensification scheme based on the approach of Fleurent and Glover but specialized for the GRASP for JSP. Other enhancements to GRASP that make use of historical information include path relinking [19] and reactive GRASP [23].

The idea in the approach of Fleurent and Glover is to maintain a set $\mathcal{E}$ of up to q diverse elite solutions and use this set to guide the construction phase of GRASP. A schedule found in any GRASP iteration is a candidate to be included in the elite set if its makespan is less than the largest makespan of the elite schedules. A candidate schedule is made elite if its makespan is smaller than the makespan of the best elite schedule (the elite schedule having the smallest makespan) or it is sufficiently different from all elite schedules. The elite set of schedules is initially empty and its worst makespan is arbitrarily set to ∞. This is kept set to ∞ until the q elite schedules are included in the elite set. Once the elite set has q schedules and a new schedule is to be added to the elite set, the elite schedule having the largest makespan is removed from the elite set.

One way to measure the difference between two schedules is to compare the order of jobs processed at each machine. Another approach is to compare start times of each operation in each schedule. In the GRASP described in this paper, the latter approach is used because of its ease of implementation. A measure of non-similarity of two schedules is the number of different start times in both schedules. Let $\Delta(A, B)$ be the number of operations having different start times in schedules A and B and let δ be a parameter used to differentiate diverse schedules from similar schedules. Furthermore, let $\mathcal{G}$ denote a GRASP schedule that is a candidate to become elite. For all elite schedules $\mathcal{E}_i$, $i = 1, \dots, q$, if $\Delta(\mathcal{G}, \mathcal{E}_i) \geq \delta \cdot |\mathcal{J}| \cdot |\mathcal{M}|$, schedule $\mathcal{G}$ is made elite because it is sufficiently different from all elite schedules.

We next show how the set of elite solutions can be used to guide the search procedure in the construction phase. As in Fleurent and Glover [14], the idea is to bias the construction procedure to reinforce good characteristics of elite solutions. Our implementation of this concept differs slightly from the implementation of Fleurent and Glover. Define $t(\mathcal{S}, \sigma)$ to be the start time of operation σ in schedule $\mathcal{S}$ and $\mathcal{S}_p$ to be the partial schedule under construction. For $\sigma \in$ RCL, the *intensity* index is defined as

$$\mathcal{I}(\sigma) = \sum_{s \in \mathcal{H}} \frac{C_{\max}(\mathcal{E}^*)}{C_{\max}(\mathcal{S})},$$

where $C_{\max}(\mathcal{E}^*)$ is the makespan of the best elite solution $\mathcal{E}^*$ and $C_{\max}(\mathcal{S})$ is the makespan of schedule $\mathcal{S}$, and

$$\mathcal{H} = \{\mathcal{S} \in \mathcal{E} \mid t(\mathcal{S}, \sigma) = t(\mathcal{S}_p, \sigma), \text{for all } \sigma \in \text{RCL}\},$$

is the set of elite schedules with operation σ having identical start time as operation σ of the partial schedule under construction. It is easy to see that the intensity of operation σ increases with the number of elite schedules having identical start times for this operation.

As discussed previously, the intensity function is used to alter the greedy heuristic function used in the construction phase. To accomplish this, define the new heuristic function, for all $\sigma \in$ RCL, to be

$$h'(\sigma) = \lambda \frac{\underline{h}}{h(\sigma)} + \mathcal{I}(\sigma), \tag{3}$$

where λ is an adjustable parameter used to take into account the fact that $0 \leq \underline{h}/h(\sigma) \leq 1$ and $0 \leq \mathcal{I}(\sigma) \leq q$, where q is the number of elite schedules, and as defined previously, $\underline{h} = h(\underline{\sigma})$.

To give equal emphasis to $\underline{h}/h(\sigma)$ and $\mathcal{I}(\sigma)$, one would set $\lambda = q$. A strategy for setting the values of λ, suggested by Fleurent and Glover, is to give more emphasis to $\underline{h}/h(\sigma)$ in the early stages of the run and gradually shift the emphasis towards the intensity function. In this implementation, we use a scheme based on the variance of the makespan obtained for the last block of k GRASP iterations. If the variance increases, the λ parameter increases by 10. Otherwise, if the variance decreases, the λ parameter decreases by 10. For the first block of $2k$ iterations, we take $\lambda = 100q$.

Having defined the new heuristic function, we are ready to describe the procedure used for selecting an element from the restricted candidate list. Initially, the RCL is set up in the same manner as described in Subsection 3.1. To select the next operation to be scheduled from the RCL, the new heuristic function is applied to all operations in the RCL and the operations are ranked according to their new heuristic function values. The probability of selecting each RCL operation is evaluated by (2) and an operation is sampled from the RCL according to this probability distribution.

5. Proximate Optimality Principle

In the construction phase of GRASP, schedules are built one operation at a time. The Proximate Optimality Principle (POP) [17], applied to this scheduling problem, states that good solutions of partial schedules with v operations are close to good solutions of partial schedules with $v + 1$ operations. An error in scheduling an operation early in the construction process may lead to other errors in scheduling operations that follow. The standard framework of GRASP corrects these erroneous steps with a local search procedure when the partial schedule is complete. However, due to the nature of local search procedures (such as 2-exchange), it may be difficult to identify and disassemble the erroneous decisions.

Fleurent and Glover [14] introduced the POP concept within the framework of a constructive multi-start method for the QAP. Periodically, they execute a local search procedure on a partial assignment (a restricted QAP problem) with the objective of reevaluating the assignments that were previously selected in the construction phase. In the context of job shop scheduling, the idea is similar, i.e. after a number of operations have been scheduled, the local search algorithm is executed on the partial schedule with the objective of reducing the makespan of the partial schedule.

To implement POP for the JSP, we use a slightly modified disjunctive graph based local search. Let $\mathcal{S}_p$ be the partial schedule and $\mathcal{O}_p$ the corresponding operations of $\mathcal{S}_p$. The disjunctive graph is built using only already-scheduled operations

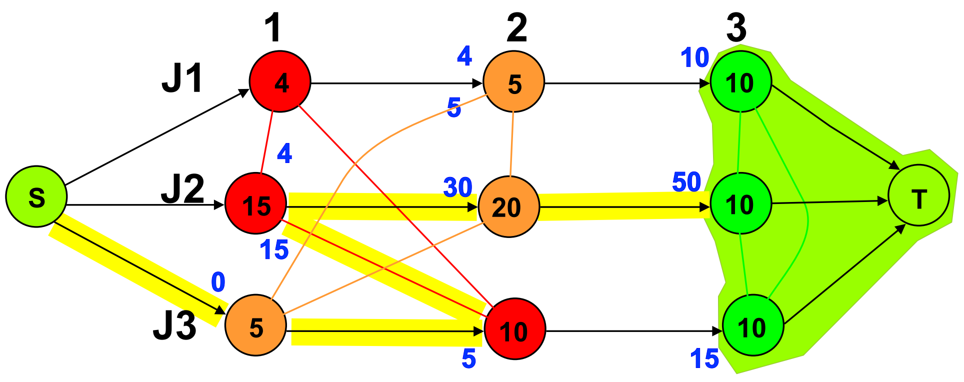

FIGURE 5. POP: Critical path of a partial schedule with a makespan of 50.

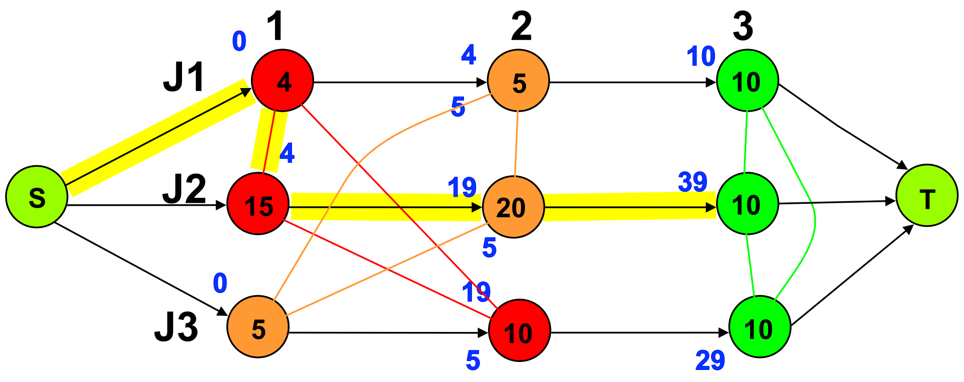

FIGURE 6. POP: Recomputing critical path and the partial schedule with a makespan of 39.

(from set $\mathcal{O}_p$), while the remaining operations are grouped in the sink node, as illustrated in Figure 5. In this way, only operations that have been already scheduled are allowed to be swapped in the local search. For example, in Figure 6, the second operation of job 2 was exchanged with the first operation of job 1, resulting in an improved in the makespan of the schedule under construction. After executing POP in the partial schedule, we continue scheduling the yet unscheduled operations in the construction phase.

Because local search can be computationally demanding, POP cannot be applied at each iteration of the construction procedure. In the implementation used in this paper, a parameter `freq` is used to determine when POP is applied. This parameter indicates the frequency of application of POP. For example, `freq` = 40 forces POP to be applied after 40% and 80% of the operations have been scheduled in the construction phase.

6. Computational experiments

To illustrate the effectiveness and performance of an implementation of the GRASP described in this paper, we consider 66 instances from five classes of standard JSP test problems: `abz`, `car`, `la`, `mt`, and `orb`. Problem dimensions vary

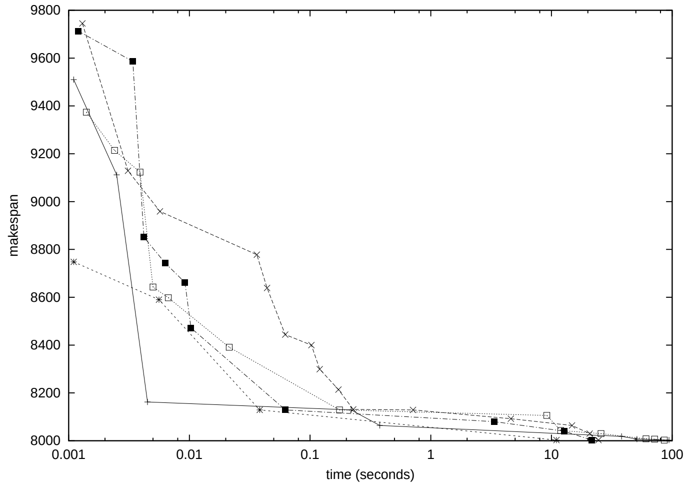

FIGURE 7. Problem `car4`: Incumbent makespan as a function of CPU time for five independent runs of GRASP. Runs stop when solution with a makespan of 8003 is found.

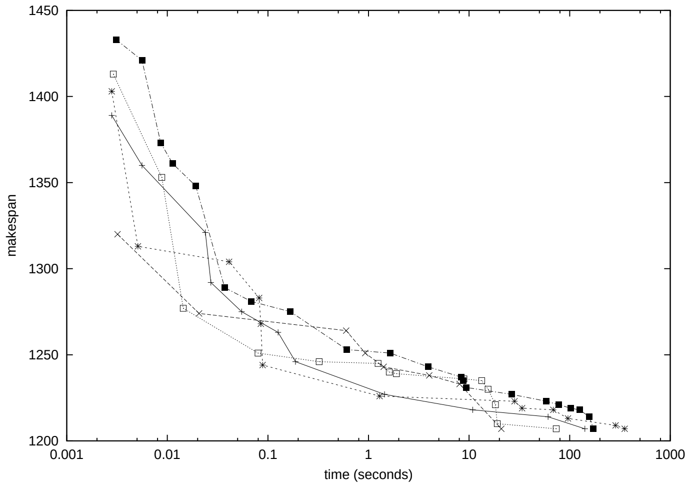

FIGURE 8. Problem `la15`: Incumbent makespan as a function of CPU time for five independent runs of GRASP. Runs stop when solution with a makespan of 1207 is found.

from 6 to 30 jobs and 4 to 20 machines. All test instances were downloaded from Beasley's OR-Library [4], `http://mscmga.ms.ic.ac.uk`.

The experiments were done on a Silicon Graphics Challenge computer (196 MHz MIPS R10000 processor). The codes were compiled with the SGI MIPSpro F77 compiler using flags `-O3 -static`. The CPU times are measured with the system routine `etime`.

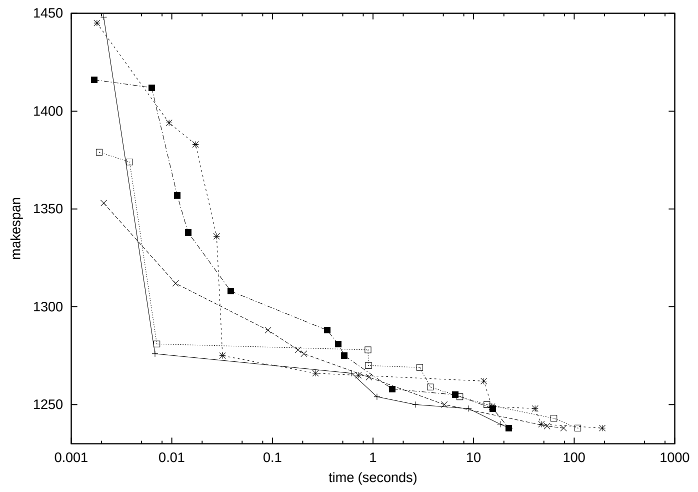

Figure 9. Problem `abz5`: Incumbent makespan as a function of CPU time for five independent runs of GRASP. Runs stop when solution with a makespan of 1238 is found.

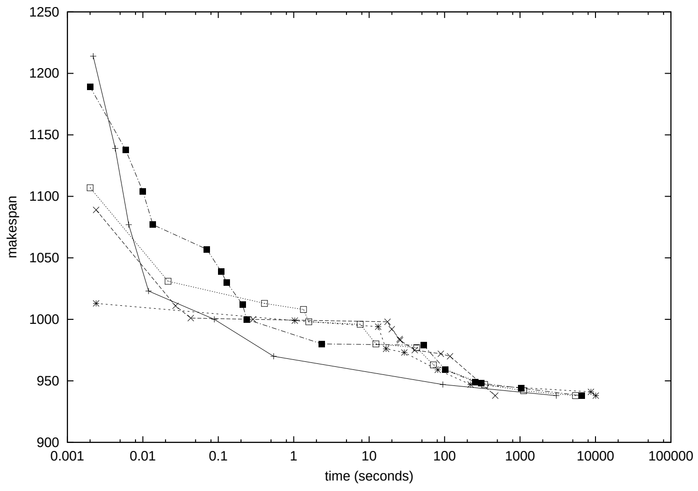

Figure 10. Problem `mt10`: Incumbent makespan as a function of CPU time for five independent runs of GRASP. Runs stop when solution with a makespan of 938 is found.

Extensive testing of the code, which we will not discuss here, determined a suitable choice of parameters. In general, we observed that the GRASP with intensification did better than the GRASP without intensification. An intensification scheme with an elite set of 30 solutions and a discrimination threshold value $\delta = 0.8$ was used. The variance of the makespans is computed every $k = 100$ iterations and the intensification parameter λ is adjusted as described in Section 4. As well, we observed that the GRASP with POP local search did better than the GRASP without POP. Consequently, we opted to experiment with a GRASP having POP. POP

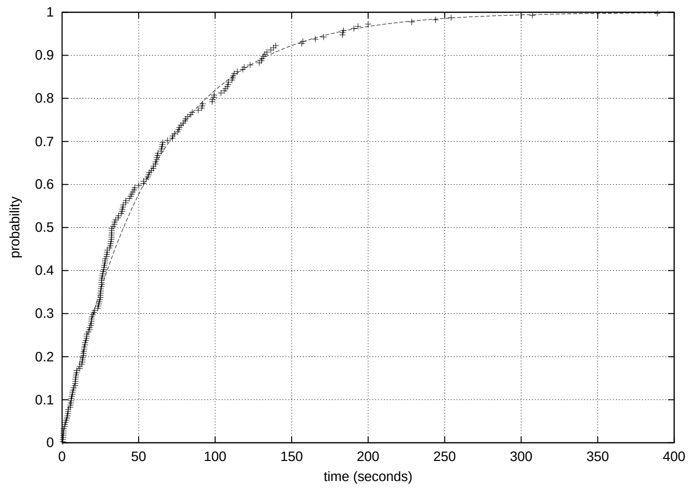

FIGURE 11. Problem `la27`: Superimposed empirical and theoretical distributions of time to sub-optimal within 10% of best known solution. Theoretical probability distribution is $F(t) = 1 - e^{-(t+0.649)/58.8688}$. Plot produced with data from 200 independent runs of GRASP.

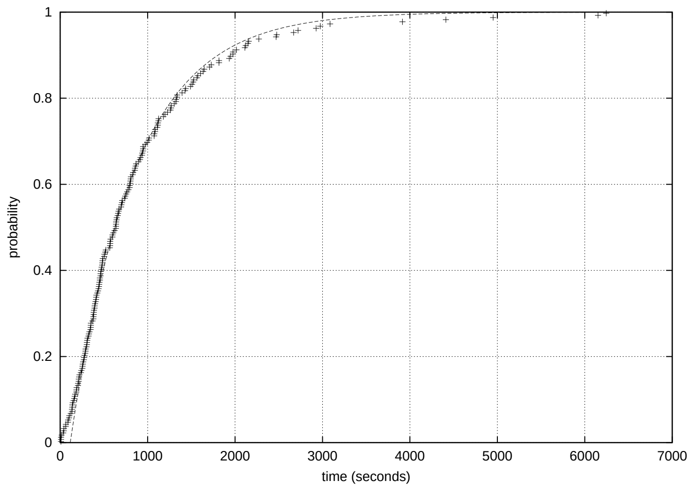

FIGURE 12. Problem `orb7`: Superimposed empirical and theoretical distributions of time to sub-optimal within 1% of best known solution. Theoretical probability distribution is $F(t) = 1 - e^{-(t-114.875)/733.5056}$. Plot produced with data from 200 independent runs of GRASP.

was activated with parameter `freq` $= 40$. In these experiments, the RCL parameter α is selected at random from the uniform distribution in the interval $[0, 1]$ and is fixed during each GRASP iteration. In general, we observed that the linear bias function was the best among the bias functions. Accordingly, a linear bias function was used for selecting candidates from the RCL.

Our objective with these experiments was to observe the general behavior of the algorithm, as well as learn how the algorithm performs on a standard set of test problems, i.e. how close to the best known solution (BKS) can the algorithm come.

Figures 7 – 10 show the general behavior of the algorithm on test problems `car4`, `la15`, `abz5`, and `mt10`, respectively. These figures show the best makespan (incumbent) as a function of CPU time for five runs of GRASP (each using a distinct initial random number generator seed). The stopping criterion on these runs was solution quality. The runs terminated after the algorithm produced for the first time the best known GRASP solution. For the first two instances, these solutions are optimal, while for `abz5` the best known GRASP solution is about 0.3% off of the optimal and for `mt10` the best known GRASP solution is about 1% off of the optimal. In these figures, one can observe the wide distribution of CPU times.

As was shown by Aiex, Resende, and Ribeiro [2], many GRASP implementations have CPU time to find a given target solution that fits a two-parameter exponential distribution. This also appears to be the case for the GRASP for job shop scheduling described in this paper. Figures 11 and 12 show superimposed empirical and theoretical distributions of solution time to a target solution within 10% of the BKS for `la27` and within 1% of the BKS for `orb7`, respectively. Each figure is generated with data from 200 independent runs of GRASP (using distinct initial random number generator seeds). The parameters of the two-parameter exponential distribution were estimated using the methodology described in [2]. Figure 11 shows, for example, that the probability of finding a solution within 10% of the BKS for `la27` in less than 50 seconds on an SGI 196-MHz R10000 processor is about 60%, while with about 80% probability one expects to find such a solution in less than 100 seconds.

To learn how the algorithm performs with respect to solution quality, we ran the implementation extensively on all of the test problems. The number of GRASP iterations was often in the millions (accomplished by restarting the procedure with different initial random number generator seeds). Tables 1 and 2 report these results. Each table lists, for each test problem, its name, dimension (number of jobs and number of machines), the total number of GRASP iterations executed, the CPU time in SGI 196-MHz R10000 seconds, the value of the best known solution (BKS), the value of the best solution found by GRASP, and the percentual relative error of the GRASP solution with respect to the BKS.

Of the 66 instances, GRASP found the BKS in 31 cases (47%). It was within 0.5% of the BKS in 38 instances (58%). In 44 cases (67%), the GRASP solution was within 1% of the BKS, and in 49 cases (74%), it was within 2% if the BKS. In 59 instances (89%) the GRASP solution was within 5% of the BKS, while for all cases the GRASP solution was within 10% of the BKS.

Tables 3 and 4 summarize these results by problem class. Table 3 shows, for each problem class, its name, the sum of the BKS values, the sum of the best GRASP solution values, and the relative error of the sum of the best GRASP solution values with respect to the sum of the BKS values. Table 4 shows, for each problem class, its name, the percentage of instances for which a GRASP solution within 0.5%, 1%, 2%, 5%, and 10% of the BKS was produced. From these tables, one can conclude that the easiest classes are `car` and `mt`, for which all GRASP solutions are within 1% of the BKS, while the most difficult are `orb`, `la`, and `abz`. Problem class `abz`

TABLE 1. Experimental results on problem classes `abz`, `car`, `mt`, and `orb`. Table shows problem name, problem dimension (jobs and machines), total number of GRASP iterations performed, CPU time per 1000 GRASP iterations, the best known solution (BKS), the best solution found by GRASP, and the relative percentual error of the GRASP solution with respect to the BKS.

problem	jobs	machines	iterations ($\times 10^6$)	time per 1000 iterations	BKS	GRASP solution	relative error (%)
`abz5`	10	10	20.1	0.3s	1234	1238	0.3
`abz6`	10	10	20.1	3.1s	943	947	0.4
`abz7`	15	20	20.1	17.4s	667	723	8.4
`abz8`	15	20	20.1	18.2s	670	729	8.8
`abz9`	15	20	20.1	17.1s	691	758	9.7
`car1`	11	5	0.1	1.4s	7038	7038	0.0
`car2`	13	4	0.1	1.5s	7166	7166	0.0
`car3`	12	5	240.1	1.5s	7312	7366	0.7
`car4`	14	4	0.1	1.6s	8003	8003	0.0
`car5`	10	6	20.1	1.3s	7702	7702	0.0
`car6`	8	9	10.1	1.5s	8313	8313	0.0
`car7`	7	7	0.1	1.0s	6558	6558	0.0
`car8`	7	7	0.1	1.0s	8264	8264	0.0
`mt06`	6	6	0.1	0.7s	55	55	0.0
`mt10`	10	10	90.1	2.9s	930	938	0.9
`mt20`	20	5	90.1	4.3s	1165	1169	0.3
`orb1`	10	10	40.1	2.9s	1059	1070	1.0
`orb2`	10	10	40.1	3.8s	888	889	0.1
`orb3`	10	10	40.1	3.1s	1005	1021	1.6
`orb4`	10	10	40.1	3.1s	1005	1031	2.6
`orb5`	10	10	40.1	2.8s	887	891	0.5
`orb6`	10	10	40.1	3.1s	1010	1013	0.3
`orb7`	10	10	10.1	3.2s	397	397	0.0
`orb8`	10	10	40.1	3.1s	899	909	1.1
`orb9`	10	10	40.1	3.1s	934	945	1.2
`orb10`	10	10	40.1	2.9s	944	953	1.0

was the one for which GRASP did worst, achieving an overall relative error of 4.52% with respect to the BKS. On the other two "hard" classes, the overall relative errors were much lower, 1.01% and 1.74%.

7. CONCLUDING REMARKS

We describe a new algorithm for finding approximate solutions to the job shop scheduling problem. The algorithm is a greedy randomized adaptive search procedure (GRASP) with an intensification-enhanced construction phase which also makes use of the Proximate Optimality Principle (POP) to correct imperfections made in the construction phase. Local search is the standard two-exchange based on the disjunctive graph model. In general, the GRASP with intensification and POP was slightly better than a variant without those schemes.

The algorithm was evaluated on 66 standard test problems and was shown to produce optimal or near-optimal solutions on all instances. Though we verified that the time to target sub-optimal solution fits well a two-parameter exponential distribution for only two instances, we feel that this is the case in general. Hence,

TABLE 2. Experimental results on problem classes `la`. Table shows problem name, problem dimension (jobs and machines), total number of GRASP iterations performed, CPU time per 1000 GRASP iterations, the best known solution (BKS), the best solution found by GRASP, and the relative percentual error of the GRASP solution with respect to the BKS.

problem	jobs	machines	iterations ($\times 10^6$)	time per 1000 iterations	BKS	GRASP solution	relative error (%)
la01	10	5	0.1	1.4s	666	666	0.0
la02	10	5	0.1	1.4s	655	655	0.0
la03	10	5	50.1	1.3s	597	604	1.2
la04	10	5	0.1	1.3s	590	590	0.0
la05	10	5	0.1	1.3s	593	593	0.0
la06	15	5	0.1	2.4s	926	926	0.0
la07	15	5	0.1	2.5s	890	890	0.0
la08	15	5	0.1	2.4s	863	863	0.0
la09	15	5	0.1	2.9s	951	951	0.0
la10	15	5	0.1	2.5s	958	958	0.0
la11	20	5	0.1	4.1s	1222	1222	0.0
la12	20	5	0.1	3.9s	1039	1039	0.0
la13	20	5	0.1	4.3s	1150	1150	0.0
la14	20	5	0.1	3.9s	1292	1292	0.0
la15	20	5	0.1	4.1s	1207	1207	0.0
la16	10	10	50.1	3.1s	945	946	0.1
la17	10	10	20.1	3.0s	784	784	0.0
la18	10	10	20.1	2.9s	848	848	0.0
la19	10	10	10.1	3.1s	842	842	0.0
la20	10	10	50.1	3.2s	902	907	0.6
la21	15	10	50.1	6.5s	1053	1091	3.6
la22	15	10	50.1	6.3s	927	960	3.6
la23	15	10	10.1	6.5s	1032	1032	0.0
la24	15	10	10.1	6.4s	935	978	4.6
la25	15	10	10.1	6.4s	977	1028	5.2
la26	20	10	10.1	10.8s	1218	1271	4.4
la27	20	10	10.1	10.9s	1269	1320	4.0
la28	20	10	10.1	10.9s	1216	1293	6.3
la29	20	10	10.1	11.1s	1195	1293	8.2
la30	20	10	10.1	10.5s	1355	1368	1.0
la31	30	10	10.1	22.9s	1784	1784	0.0
la32	30	10	10.1	23.9s	1850	1850	0.0
la33	30	10	10.1	23.9s	1719	1719	0.0
la34	30	10	10.1	23.8s	1721	1753	1.9
la35	30	10	10.1	22.0s	1888	1888	0.0
la36	15	15	11.2	10.3s	1268	1334	5.2
la37	15	15	11.2	10.3s	1397	1457	4.3
la38	15	15	11.2	10.6s	1217	1267	4.1
la39	15	15	11.2	10.3s	1233	1290	4.6
la40	15	15	11.2	11.0s	1222	1259	3.0

as discussed in [2], this algorithm can achieve linear speed-up with a parallel implementation.

To find better solutions on some of the notoriously difficult instances, such as `abz9` and `la29`, where the algorithm found solutions about 10% off of the best known solution, an intensification scheme may be required to search the solution

Table 3. Experimental results: Overall solution quality by problem class. Sum of all best known solutions (BKS) for each class is compared with sum of best GRASP solutions. Relative error is of GRASP solution with respect to BKS.

problem	sum of BKS	sum of GRASP sol.	relative error (%)
`abz`	4205	4395	4.52
`car`	60356	60410	0.09
`mt`	2150	2162	0.56
`orb`	9028	9119	1.01
`la`	44396	45168	1.74

Table 4. Experimental results: Percentage of GRASP solutions within a tolerance of the best known solution (BKS).

	percentage of GRASP solutions within					
problem	optimal	0.5% of BKS	1% of BKS	2% of BKS	5% of BKS	10% of BKS
`abz`	0.0	40.0	40.0	40.0	40.0	100.0
`car`	87.5	87.5	100.0	100.0	100.0	100.0
`mt`	33.3	66.6	100.0	100.0	100.0	100.0
`orb`	10.0	40.0	60.0	90.0	100.0	100.0
`la`	44.4	57.5	62.5	67.5	90.0	100.0

space around the elite solutions. One way in which this can be done is via path relinking, as described by Laguna and Martí [19].

References

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Sciences*, 34:391–401, 1988.

[2] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. Technical report, Information Sciences Research Center, AT&T Labs Research, Florham Park, NJ 07932 USA, 2000.

[3] D. Applegate and W.Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3:149–156, 1991.

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

[5] J.L. Bresina. Heuristic-biased stochastic sampling. In *Proceedings of the AAAI-96*, pages 271–278, 1996.

[6] P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:105–127, 1994.

[7] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35:164–176, 1989.

[8] J. Carlier and E. Pinson. A practical use of Jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, 26:269–287, 1990.

[9] L. Davis. Job shop scheduling with genetic algorithms. In *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, pages 136–140. Morgan Kaufmann, 1985.

[10] 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.

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

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

[13] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice Hall, Englewood Cliffs, NJ, 1963.

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

[15] S. French. *Sequencing and scheduling:An introduction to the mathematics of the job-shop.* Horwood, 1982.

[16] B. Giffler and G. L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1960.

[17] F. Glover and M. Laguna. Tabu search. In C. R. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–141. Blackwell Scientific Publications, Oxford, 1993.

[18] A. S. Jain and S. Meeran. A state-of-the-art review of job-shop scheduling techniques. Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, 1998.

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

[20] J. K. Lenstra and A. H. G. Rinnooy Kan. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.

[21] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42:797–813, 1996.

[22] E. Pinson. The job shop scheduling problem: A concise survey and some recent developments. In P. Chrétienne, E.G. Coffman Jr., J.K. Lenstra, and Z. Liu, editors, *Scheduling theory and its application*, pages 277–293. John Wiley and Sons, 1995.

[23] M. Prais and C.C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. Technical report, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, RJ 22453-900 Brazil, January 1998.

[24] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec constraints disjoinctives, 1964.

[25] E. D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6:108–117, 1994.

[26] R. J. M. Vaessens, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by local search. *INFORMS Journal on Computing*, 8:302–317, 1996.

[27] P. J. M. Van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.

(S. Binato) Electrical Energy Research Center (CEPEL), P.O. Box 68007, Rio de Janeiro, RJ 21944-970 Brazil.
E-mail address, S. Binato: `silvio@cepel.br`

(W.J. Hery) Bell Laboratories, Lucent Technologies, Whippanny, NJ 07981 USA.
E-mail address, W.J. Hery: `wjh@lucent.com`

(D.M. Loewenstern) Department of Computer Science, Rutgers University, Piscataway, NJ 08855 USA.
E-mail address, D.M. Loewenstern: `loewenst@paul.rutgers.edu`

(M.G.C. Resende) Information Sciences Research, AT&T Labs Research, Florham Park, NJ 07932 USA.
E-mail address, M.G.C. Resende: `mgcr@research.att.com`