A Simple Optimized Search Heuristic for the Job-Shop Scheduling Problem

Fernandes, S. and **Lourenço, H.R.** (2007), A Simple Optimized Search Heuristic for the Job-Shop Scheduling Problem. In Proceeding of the V Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, MAEB'2007 F. Rodriguez, B. Mélian, J.A. Moreno, J.M. Moreno (Eds.) Tenerife, Spain, February 14-16, pp. 763-770.

ISBN 978-84-690-3470-5.

A Simple Optimized Search Heuristic for the Job-Shop Scheduling Problem

Susana Fernandes[1], Helena R. Lourenço[2]

Abstract-- **This paper presents a simple algorithm for the job shop scheduling problem that combines GRASP (a heuristic with a local search phase), with a branch-and-bound exact method of integer programming. The proposed method is compared with similar approaches and leads to better results in terms of solution quality and computational times.**

Keywords—**job-shop scheduling, hybrid metaheuristic, GRASP, branch-and-bound.**

I. Introduction

The job-shop scheduling problem has been known to the operations research community since the early 50's (Jain and Meeran 1999). It is considered a particularly hard combinatorial optimization problem of the NP-hard class (Garey and Johnson 1979) and it has numerous practical applications; which makes it an excellent test problem for the quality of new scheduling algorithms. These are main reasons for the vast bibliography on both exact and heuristic procedures applied to this scheduling problem. The paper of Jain and Meeran. (1999) includes an exhaustive survey not only of the evolution of the definition of the problem, but also of all the techniques applied to it.

Recently a new class of procedures that combine local search based (meta) heuristics and exact algorithms have been developed, we denominate them Optimized Search Heuristics (OSH), (Fernandes and Lourenço 2006). This paper presents a simple OSH procedure for the job shop scheduling problem that combines a GRASP algorithm with a branch-and-bound method.

We first introduce the job-shop scheduling problem. We present a short review of existent OSH methods applied to this problem and proceed describing the procedure developed. Computational results are presented along with comparisons to other procedures.

II. The Job-Shop Scheduling Problem

The job-shop scheduling problem considers a set of jobs to be processed in a set of machines. Each job is defined by an ordered set of operations and each operation is assigned to a machine with a predefined constant processing time. Preemption is not allowed when processing operations. The order of the operations within the jobs and its correspondent machines are fixed a priory and independent from job to job. To solve the problem we need to find a sequence of operations in each machine satisfying some constrains and optimizing some objective function. It is assumed that two consecutive operations of the same job are assigned to different machines, each machine can only process one operation at a time, and that different machines can not process the same job simultaneously. We will adopt the maximum of the completion time of all jobs – the makespan – as the objective function.

Formally let $O = \{0,\ldots,o+1\}$ be the set of operations with 0 and o+1 being the dummy operations representing the start and end of all jobs, respectively. Let M be the set of machines, A the set of pairs of consecutive operations of each job and E_k the set of all possible pairs of operations processed by machine k, with $k \in M$. We define $p_i > 0$ as the constant processing time of operation i and t_i is the variable representing the starting time of operation i. The mathematical formulation for the job shop scheduling problem is as follows:

$$\min \quad t_{o+1}$$

s.t.

$$t_j - t_i \geq p_i \qquad (i,j) \in A \qquad \textbf{(1)}$$

$$t_i \geq 0 \qquad i \in O \qquad \textbf{(2)}$$

$$t_j - t_i \geq p_i \vee t_i - t_j \geq p_j \qquad (i,j) \in E_k, k \in M \qquad \textbf{(3)}$$

The job shop scheduling problem is usually represented by a disjunctive graph (Roy and Sussman 1964) $G = (O, A, E)$. Where O is the node set, corresponding to the set of operations. A is the set of arcs between consecutive operations of the same job, and E is the set of edges between operations processed by the same machine. Each node i has weight p_i, with $p_0 = p_{o+1} = 0$. There is a subset of nodes O_k and a subset of edges E_k for each machine that together form the disjunctive clique $C_k = (O_k, E_k)$ of graph G. For every node j of $O/\{0, o+1\}$ there are unique nodes i and l such that arcs (i,j) and (j,l) are elements of A. Node i

[1] Universidade do Algarve, Faro Portugal. E-mail: sfer@ualg.pt
[2] Univertitat Pompeu Fabra, Barcelona, Spain. E-mail: helena.ramalhinho@upf.edu

is called the job predecessor of node j - $jp(j)$ and l is the job successor of j - $js(j)$.

Finding a solution to the job-shop scheduling problem means replacing every edge of the respective graph with a directed arc, constructing an acyclic directed graph $D_S = (O, A \cup S)$. Graph $D = (O, A)$ is obtained from G removing all edges and $S = \bigcup_k S_k$ corresponds to an acyclic union of sequences of operations for each machine k (this implies that a solution can be built sequencing one machine at a time).

The optimal solution is the one represented by the graph D_S having the critical path from 0 to $o+1$ with the smallest length.

For any given solution, the operation processed immediately before operation i in the same machine is called the machine predecessor of i - $mp(i)$; analogously $ms(i)$ is the operation that immediately succeeds i at the same machine.

III. Review of Optimized Search Heuristics

In the literature we can find a few works combining metaheuristics with exact algorithms applied to the job shop scheduling problem. They vary not only on the procedures combined but also in the way of combining them.

Chen et al. (1993) and Denzinger and Offermann (1999) design parallel algorithms that use asynchronous agents information to build solutions; some of these agents are genetic algorithms, others are branch-and-bound algorithms.

Tamura et al. (1994) design a genetic algorithm where the fitness of each individual, whose chromosomes represent each variable of the integer programming formulation, is the bound obtained solving lagrangian relaxations.

The works of Adams et al. (1988), Applegate and Cook (1991), Caseau and Laburthe (1995) and Balas and Vazacopoulos (1998) all use an exact algorithm to solve a sub problem within a local search heuristic for the job shop scheduling. Caseau and Laburthe (1995) build a local search where the neighborhood structure is defined by a subproblem that is exactly solved using constraint programming. Applegate and Cook (1991) develop the shuffle heuristic. At each step of the local search the processing orders of the jobs on a small number of machines is fixed, and a branch-and-bound algorithm completes the schedule. The shifting bottleneck heuristic, due to Adams, Balas and Zawack (1988), is an iterated local search with a construction heuristic that uses a branch-and-bound to solve the subproblems of one machine with release and due dates. Balas and Vazacopoulos (1998) work with the shifting bottleneck heuristic and design a guided local search, over a tree search structure, that reconstructs partially destroyed solutions.

Lourenço (1995) and Lourenço and Zwijnenburg (1996) use branch-and-bound algorithms to strategically guide an iterated local search, and also a tabu search algorithm. The diversification of the search is achieved using the optimal solution of subproblems (of one or two machines) to determine the perturbation forced on the incumbent solution.

In the work of Schaal et al. (1999) an interior point method generates initial solutions of the linear relaxation. A genetic algorithm finds integer solutions. A cut is generated based on the integer solutions found and the interior point method is applied again to diversify the search. This procedure is defined for the generalized job shop problem.

The interesting work done by Danna Rothberg and Le Pape (2005) "applies the spirit of metaheuristics" in an exact algorithm. Within each node of a branch-and-cut tree, the solution of the linear relaxation is used to define the neighborhood of the current best feasible solution. The local search consists in solving the restricted MIP problem defined by the neighborhood.

IV. Optimized Search Heuristic – GRASP_B&B

We developed a simple optimized search heuristic that combines a GRASP algorithm with a branch-and-bound method. Here the branch-and-bound is used within the GRASP to solve subproblems of one machine scheduling.

GRASP (Feo and Resende 1995) means "greedy randomized adaptive search procedure". It is an iterative process where each iteration consists of two steps: a randomized building step of a greedy nature and a local search step. At the building phase, a feasible solution is constructed by joining one element at a time. The element is defined specifically for each problem. Each element is evaluated by a heuristic function and incorporated (or not) in a restricted candidate list (RCL) according to its evaluation. The element to join the solution is chosen randomly from the RCL.

Each time a new element is added to the partial solution the algorithm proceeds with the local search step. The current solution is updated by the local optimum and this process of two steps is repeated until the solution is complete.

A. Building Block

We define the sequence of operations at each machine as the elements to join the solution, and their makespan ($\max(t_i + p_i), i \in O_k, k \in M$) as the greedy function to evaluate them. In order to build the restricted candidate list we find the optimal solution for the one machine problems of machines not yet scheduled, and identify the best $(\underline{f})$ and

worse $(\overline{f})$ makespans. A machine k is included in the RCL if $f(x_k) \geq \overline{f} - \alpha(\overline{f} - \underline{f})$, where $f(x_k)$ is the makespan of machine k and α is a uniform random number in $(0,1)$. This semi-greedy randomised procedure is biased towards the machine with the highest makespan, the bottleneck machine, in the sense that machines with low values of makespan have less probability of being included in the restricted candidate list.

SemiGreedy (K)
(1) $\alpha := Random(0,1)$
(2) $\overline{f} := \max\{f(x_k), k \in K\}$
(3) $\underline{f} := \min\{f(x_k), k \in K\}$
(4) $RCL = \{\ \}$
(5) foreach $k \in K$
(6) if $f(x_k) \geq \overline{f} - \alpha(\overline{f} - \underline{f})$
(7) $RCL := RCL \cup \{k\}$
(8) return RandomChoice(RCL)

To solve the one machine scheduling problems we use the branch-and-bound algorithm of Carlier (1982). The objective function of the algorithm is to minimize the completion time of all jobs and it assumes that there are three values associated with every job j to be scheduled at the machine: the processing time (p_j), a release date (r_j) and an amount of time (q_j) that the job stays in the system after being processed.

At each node of the branch-and-bound tree the upper bound is computed using the algorithm of Schrage (1970). This algorithm gives priority to higher values of the tails (q_j) when scheduling released jobs. We break ties preferring jobs with bigger processing times.

The lower bound defined by Carlier (1982) is based on a critical path (the one with more jobs) of the solution found by the algorithm of Schrage (1970). The value of the solution with preemption is used to strengthen this lower bound. We introduce a slight modification, forcing the lower bound of a node never to be smaller than the one of its father in the tree.

The algorithm of Carlier (1982) uses some proven properties of the one machine scheduling problem to define the branching strategy, and also to reduce the number of inspected nodes of the branch-and-bound tree.

When applying the algorithm to problems with 50 or more jobs, we observed that a lot of time was spent inspecting nodes of the tree, after having already found the optimal solution. So we introduced a condition restricting the number of nodes of the tree: the algorithm is stopped if there have been inspected more then n^3 nodes after the last reduction of the difference between the upper and lower bound of the tree (n is the number of jobs).

Considering the job-shop problem and its disjunctive graph representation, the release date of each operation i - (r_i) is defined as the longest path from the beginning to i, and its tail (q_i) as the longest path from i to the end, without the processing time of i.

At the first iteration we consider the graph $D = (O, A)$ (without the edges connecting operations that share the same machine) to compute release dates and tails. Incorporating a new machine in the solution means adding to the graph the arcs representing the sequence of operations in that machine. We then update the makespan of the partial solution and the release dates and tails of unscheduled operations, using the algorithm of Taillard (1994).

B. Local Search

In order to build a simple local search algorithm we need to design a neighborhood structure (defined by moves between solutions), the way to inspect the neighborhood of a given solution, and a procedure to evaluate the quality of each solution. It is said that a solution B is a neighbor of a solution A if we can achieve B by performing a move in A.

We use a neighborhood structure very similar to the NB neighborhood of Dell'Amico and Trubian (1993) and the one of Balas and Vazacopoulos (1998). To describe the moves that define this neighborhood we use the notion of blocks of critical operations. A block of critical operations is a maximal ordered set of consecutive operations of a critical path (in the disjunctive graph that represents the solution), sharing the same machine. Let $L(i, j)$ denote the length of the critical path from node i to node j. Borrowing the name used by Balas and Vazacopoulos (1998) we speak of forward and backward moves over forward and backward critical pairs of operations.

Two operations u and v form a forward critical pair (u,v) if:

a) they both belong to the same block;

b) v is the last operation of the block;

c) operation $js(v)$ also belongs to the same critical path;

d) the length of the critical path from v to $o+1$ is not less than the length of the critical path from $js(u)$ to $0+1$ ($L(v,o+1) \geq L(js(u),o+1)$).

Two operations u and v form a backward critical pair (u,v) if:

a) they both belong to the same block;

b) u is the first operation of the block;

c) operation $jp(u)$ also belongs to the same critical path;

d) the length of the critical path from 0 to u, including the processing time of u, is not less than the length of the critical path from 0 to $jp(v)$, including the processing time of $jp(v)$ ($L(0,u)+p_u \geq L(0, jp(v))+p_{jp(v)}$).

Conditions d) are included to guarantee that all moves lead to feasible solutions (Balas and Vazacopoulos 1998).

A forward move is executed by moving operation u to be processed immediately after operation v. A backward move is executed by moving operation v to be processed immediately before operation u.

When inspecting the neighborhood ($N(x,M_k)$) of a given solution x with M_k machines already scheduled, we stop whenever we find a neighbor with a best evaluation value then the makespan of x.

To evaluate the quality of a neighbor of a solution x, produced by a move over a critical pair (u,v), we need only to compute the length of all the longest paths through the operations that were between u and v in the critical path of solution x. This evaluation is computed using the algorithm of Balas and Vazacopoulos (1998), which is a variation of the one of Taillard (1994) for a subset of arcs.

LocalSearch $(x, f(x), M_0)$

(1) $s := neighbor(x, f(x), M_0)$
(2) while $s \neq x$
(3) $\quad x := s$
(4) $\quad s := neighbor(x, f(x), M_0)$
(5) return s

Neighbor $(x, f(x), M_0)$

(1) foreach $s \in N(x, M_0)$
(2) $f(s) := evaluation(move(x \rightarrow s))$
(3) $\quad$ if $f(s) < f(x)$
(4) $\quad\quad$ return s
(5) return x

C. GRASP_B&B

We named the procedure GRASP_B&B. Let runs be the total number of runs, M the set of machines of the instance and $f(x)$ the makespan of a solution x. The procedure can be generally described by the pseudo-code in the following figure:

GRASP_B&B (runs)

(1) $M := \{1, \cdots, m\}$
(2) for $r = 1$ to runs
(3) $\quad x := \{\ \}$
(4) $\quad K := M$
(5) $\quad$ while $K \neq \{\ \}$
(6) $\quad\quad$ foreach $k \in K$
(7) $\quad\quad\quad x_k := CARLIER\_B \& B(k)$
(8) $\quad\quad k^* := SEMIGREEDY(K)$
(9) $\quad\quad x := x \cup x_{k^*}$
(10) $\quad\quad f(x) := TAILLARD(x)$
(11) $\quad\quad K := K \setminus \{k^*\}$
(12) $\quad\quad$ if $|K| < |M| - 1$
(13) $\quad\quad\quad x := LOCALSEARCH(x, M \setminus K)$
(14) $\quad\quad$ if x^* not initialized or $f(x) < f^*$
(15) $\quad\quad\quad x^* := x$
(16) $\quad\quad\quad f^* := f(x)$
(17) $\quad$ return x^*

This metaheuristic has only one parameter to be defined: the number of runs to perform (line (2)). The step of line (8) is the only one using randomness. When applied to an instance with m machines, in each run of the metaheuristic, the branch-and-bound algorithm is called $m \times (m+1)/2$ times (line (7)); the local search is executed $m-1$ times (lines (12) and (13)); the procedure semi-greedy (line (8)) and the algorithm of Taillard (line (10)) are executed m times.

D. Computacional Results

We have tested the algorithm GRASP_B&B on the benchmark instances abz5-9 (Adams et al. 1988), ft6, ft10, ft20 (Fisher and Thompson 1963), la01-40 (Lawrence 1984), orb01-10 (Applegate and Cook 1991), swv01-20 (Storer et al. 1992), ta01-70 (Taillard 1993) and yn1-4 (Yamada and Nakano 1992). Because of space limitations, in this work we will only present the results for ft6, ft10, ft20 (Fisher and Thompson 1963), la01-40 (Lawrence 1984) and orb01-10 (Applegate and Cook 1991), in Tables I, II and III respectively.

The tables have the following structure: in each line it is presented the name of the instance, the number of jobs and the number of machines of the instance (n*m), the best value (min) obtained, the percentage over the lower bound and the time to the best solution found (btime), in seconds. The total time of all runs (ttime) will be presented in next tables. We gathered the values of the lower bounds from the paper of Jain and Meeran (1999) and the

papers of Nowicki and Smutnicki (1996, 2002, 2005). Within brackets, next to the best known value (UB), is the percentage of relative error to the upper bound calculated as follows:

$$RE_{UB}(x) = 100\% \times \frac{x - UB}{UB}$$

The algorithm has been run 100 times for each instance on a Pentium 4 CPU 2.80 GHz and coded in C.

Whenever the values are not worse than the best known upper bound, we present them in bold. Although this is a very simple (and fast) algorithm, it happens in 22 of the 152 instances used in this study.

TABLA I
RESULTS OF THE **FT** INSTANCES

name	n*m	min	btime (s)
ft06	6*6	**55** (0.00)	0.1274
ft10	10*10	970 (4.30)	0.5800
ft20	20*5	1283 (10.13)	0.0094

TABLA II
RESULTS OF THE **LA** INSTANCES

name	n*m	min	btime (s)
la01	10*5	**666** (0.00)	0.0017
la02	10*5	667 (1.83)	0.0437
la03	10*5	605 (1.34)	0.0066
la04	10*5	607 (2.88)	0.0051
la05	10*5	**593** (0.00)	0.0011
la06	15*5	**926** (0.00)	0.0017
la07	15*5	**890** (0.00)	0.0020
la08	15*5	**863** (0.00)	0.0149
la09	15*5	**951** (0.00)	0.0028
la10	15*5	**958** (0.00)	0.0014
la11	20*5	**1222** (0.00)	0.0027
la12	20*5	**1039** (0.00)	0.0027
la13	20*5	**1150** (0.00)	0.0038
la14	20*5	**1292** (0.00)	0.0022
la15	20*5	**1207** (0.00)	0.0453
la16	10*10	1012 (7.09)	0.0221
la17	10*10	787 (0.38)	0.0843
la18	10*10	854 (0.71)	0.3000
la19	10*10	861 (2.26)	0.4554
la20	10*10	920 (2.00)	0.0813
la21	15*10	1092 (4.40)	0.1023
la22	15*10	955 (3.02)	0.9884
la23	15*10	1049 (1.65)	1.7388
la24	15*10	971 (3.85)	0.6270
la25	15*10	1027 (5.12)	0.5388
la26	20*10	1265 (3.86)	3.0375
la27	20*10	1308 (5.91)	0.1781
la28	20*10	1301 (6.99)	0.1500
la29	20*10	1248 (8.33)	0.8570
la30	20*10	1382 (1.99)	0.8653
la31	30*10	**1784** (0.00)	0.0702
la32	30*10	**1850** (0.00)	0.5612
la33	30*10	**1719** (0.00)	1.2650
la34	30*10	**1721** (0.00)	3.8093
la35	30*10	**1888** (0.00)	0.2844
la36	15*15	1325 (4.50)	0.0853
la37	15*15	1479 (5.87)	4.0295
la38	15*15	1274 (6.52)	0.7153
la39	15*15	1309 (6.16)	2.9835
la40	15*15	1291 (5.65)	3.5581

TABLA III
RESULTS OF THE **ORB** INTANCES

name	n*m	min	btime (s)
orb01	10*10	1145 (8.12)	0.0296
orb02	10*10	918 (3.38)	0.0953
orb03	10*10	1098 (9.25)	0.3350
orb04	10*10	1066 (6.07)	0.8213
orb05	10*10	911 (2.71)	0.1050
orb06	10*10	1050 (3.96)	0.4812
orb07	10*10	414 (4.28)	0.2764
orb08	10*10	945 (5.12)	0.3093
orb09	10*10	978 (4.71)	0.2809
orb10	10*10	991 (4.98)	0.2276

In a previous version of this algorithm (Fernandes 2002), where a slightly different branch-and-bound was only used the first time the one-machine subproblems were solved, the results achieved where worse than these ones. In that work, the neighborhood used was the one of Balas and Vazacopoulos (1998). Computational times can not be compared since then the algorithm was coded in Python which is much slower than C, and there were only used 13 of the 40 instances of Lawrence (1984) in the experiments.

GRASP_B&B is a very simple GRASP algorithm with a construction phase very similar to the one of the shifting bottleneck. Therefore we show comparison results to two other procedures design for the job shop problem; a simple GRASP procedure by Binato et al (2002) and the shifting bottleneck procedure (Adams et al. 1988), see tables IV to VIII.

The building block of the construction phase of the GRASP in (Binato et al. 2002) is a single operation of a job. They use an intensification strategy, based on a set of elite solutions, to 'direct' the random choice of a new element. The local search applied uses the neighborhood defined by exchanging two consecutive critical operations on the same machine.

In their computational results, they present the time in seconds per thousand iterations (an iteration is one building phase followed by a local search), and the thousands of iterations. For a comparison purpose we multiply these values to get the total computation time. As the tables show, our algorithm is much faster. Whenever our GRASP achieves a solution not worse than theirs, we present the respective value in bold. This happens for 25 of the 53 instances whose results we present here.

TABLA IV
COMPARISON WITH BINATO ET. AL (2002) OF THE **FT** INSTANCES

name	*GRASP_ B&B*	*ttime (s)*	*GRASP (Binato et al. 2002)*	*time (s)*
ft06	***55***	*0.1400*	*55*	*70*
ft10	*970*	*1.0000*	*938*	*261290*
ft20	*1283*	*0.4690*	*1169*	*387430*

TABLA V
COMPARISON WITH BINATO ET. AL (2002) OF THE **LA** INSTANCES

name	*GRASP_B&B*	*ttime (s)*	*GRASP (Binato et al. 2002)*	*time (s)*
la01	***666***	*0.1720*	*666*	*140*
la02	*667*	*0.1560*	*655*	*140*
la03	*605*	*0.2190*	*604*	*65130*
la04	*607*	*0.1710*	*590*	*130*
la05	***593***	*0.1100*	*593*	*130*
la06	***926***	*0.1710*	*926*	*240*
la07	***890***	*0.2030*	*890*	*250*
la08	***863***	*0.2970*	*863*	*240*
la09	***951***	*0.2810*	*951*	*290*
la10	***958***	*0.1410*	*958*	*250*
la11	***1222***	*0.2660*	*1222*	*410*
la12	***1039***	*0.2650*	*1039*	*390*
la13	***1150***	*0.3750*	*1150*	*430*
la14	***1292***	*0.2180*	*1292*	*390*
la15	***1207***	*0.9060*	*1207*	*410*
la16	*1012*	*0.7350*	*946*	*155310*
la17	*787*	*0.7660*	*784*	*60300*
la18	*854*	*0.7500*	*848*	*58290*
la19	*861*	*0.9690*	*842*	*31310*
la20	*920*	*0.8130*	*907*	*160320*
la21	*1092*	*2.0460*	*1091*	*325650*
la22	***955***	*1.7970*	*960*	*315630*
la23	*1049*	*1.8900*	*1032*	*65650*
la24	***971***	*1.8440*	*978*	*64640*
la25	***1027***	*1.7960*	*1028*	*64640*
la26	***1265***	*3.3750*	*1271*	*109080*
la27	***1308***	*3.5620*	*1320*	*110090*
la28	*1301*	*3.0000*	*1293*	*110090*
la29	***1248***	*3.2960*	*1293*	*112110*
la30	*1382*	*3.3280*	*1368*	*106050*
la31	***1784***	*7.0160*	*1784*	*231290*
la32	***1850***	*6.2350*	*1850*	*241390*
la33	***1719***	*7.9060*	*1719*	*241390*
la34	***1721***	*8.2810*	*1753*	*240380*
la35	***1888***	*5.6880*	*1888*	*222200*

la36	***1325***	*4.2650*	*1334*	*115360*
la37	*1479*	*4.7970*	*1457*	*115360*
la38	*1274*	*5.1090*	*1267*	*118720*
la39	*1309*	*4.4530*	*1290*	*115360*
la40	*1291*	*5.3910*	*1259*	*123200*

TABLA VI
COMPARISON WITH BINATO ET. AL (2002) OF THE **ORB** INSTANCES

name	*GRASP_B&B*	*ttime (s)*	*GRASP (Binato et al. 2002)*	*time (s)*
orb01	*1145*	*0.9850*	*1070*	*116290*
orb02	*918*	*0.9530*	*889*	*152380*
orb03	*1098*	*1.0150*	*1021*	*124310*
orb04	*1066*	*1.1250*	*1031*	*124310*
orb05	*911*	*0.8750*	*891*	*112280*
orb06	*1050*	*1.0460*	*1013*	*124310*
orb07	*414*	*1.0630*	*397*	*128320*
orb08	*945*	*1.0310*	*909*	*124310*
orb09	*978*	*0.9060*	*945*	*124310*
orb10	*991*	*0.8430*	*953*	*116290*

The comparison between the shifting bottleneck procedure (Adams et al. 1988) and the GRASP_B&B are presented in tables VII and VIII. The main differences between these two simple heuristics are as follows: the machine added to the solution is always the one with higher makespan; every time a new machine is included, the subproblem of each of the already scheduled machines is reoptimized, restricted to all the other scheduled machines; when the solution is complete, the cycle of reoptimizing each one-machine problem is repeated until there are no changes.

Comparing the computational times of both procedure, our GRASP is slightly faster than the shifting bottleneck for smaller instances. Given the distinct computers used in the experiments we would say that this is not meaningful, but the difference does get accentuated as the dimensions grow. Whenever GRASP_B&B achieves a solution better than the shifting bottleneck procedure, we present it's value in bold. This happens in 29 of the 48 instances whose results where compared, and in 16 of the remaining 19 instances the best value found was the same.

TABLA VII
COMPARISON WITH ADAMS ET AL.(1988) OF THE **FT** INSTANCES

name	*GRASP_B&B*	*ttime (s)*	*Shifting Bottleneck*	*time (s)*
ft06	*55*	*0.1400*	*55*	*1.5*
ft10	***970***	*1.0000*	*1015*	*10.1*
ft20	***1283***	*0.4690*	*1290*	*3.5*

TABLA VIII
COMPARISON WITH ADAMS ET AL.(1988) OF THE **LA** INSTANCES

name	*GRASP_B&B*	*ttime (s)*	*Shifting Bottleneck*	*time (s)*
la01	*666*	*0.1720*	*666*	*1.26*
la02	***667***	*0.1560*	*720*	*1.69*
la03	***605***	*0.2190*	*623*	*2.46*
la04	*607*	*0.1710*	*597*	*2.79*
la05	*593*	*0.1100*	*593*	*0.52*
la06	*926*	*0.1710*	*926*	*1.28*
la07	*890*	*0.2030*	*890*	*1.51*
la08	***863***	*0.2970*	*868*	*2.41*
la09	*951*	*0.2810*	*951*	*0.85*
la10	***958***	*0.1410*	*959*	*0.81*
la11	*1222*	*0.2660*	*1222*	*2.03*
la12	*1039*	*0.2650*	*1039*	*0.87*
la13	*1150*	*0.3750*	*1150*	*1.23*
la14	*1292*	*0.2180*	*1292*	*0.94*
la15	*1207*	*0.9060*	*1207*	*3.09*
la16	***1012***	*0.7350*	*1021*	*6.48*
la17	***787***	*0.7660*	*796*	*4.58*
la18	***854***	*0.7500*	*891*	*10.2*
la19	***861***	*0.9690*	*875*	*7.4*
la20	***920***	*0.8130*	*924*	*10.2*
la21	***1092***	*2.0460*	*1172*	*21.9*
la22	***955***	*1.7970*	*1040*	*19.2*
la23	***1049***	*1.8900*	*1061*	*24.6*
la24	***971***	*1.8440*	*1000*	*25.5*
la25	***1027***	*1.7960*	*1048*	*27.9*
la26	***1265***	*3.3750*	*1304*	*48.5*
la27	***1308***	*3.5620*	*1325*	*45.5*
la28	*1301*	*3.0000*	*1256*	*28.5*
la29	***1248***	*3.2960*	*1294*	*48*
la30	***1382***	*3.3280*	*1403*	*37.8*
la31	*1784*	*7.0160*	*1784*	*38.3*
la32	*1850*	*6.2350*	*1850*	*29.1*
la33	*1719*	*7.9060*	*1719*	*25.6*
la34	*1721*	*8.2810*	*1721*	*27.6*
la35	*1888*	*5.6880*	*1888*	*21.3*
la36	***1325***	*4.2650*	*1351*	*46.9*
la37	***1479***	*4.7970*	*1485*	*6104*
la38	***1274***	*5.1090*	*1280*	*57.5*
la39	***1309***	*4.4530*	*1321*	*71.8*
la40	***1291***	*5.3910*	*1326*	*76.7*

V. CONCLUSIONS

We have designed a very simple optimized search heuristic, the GRASP_B&B. It is intended to be a starting point for a more elaborated metaheuristic. We have compared it to other base procedures used within more complex algorithms; namely a GRASP of Binato et al. (2002), which is the base for a GRASP with path-relinking procedure

(Aiex et al. 2003), and the shifting bottleneck procedure, incorporated in the successful guided local search of Balas and Vazacopoulos (1998). The comparison to the work of Binato et al. (2002) shows that our GRASP is much faster than theirs, with differences as big as less than one second to hundreds of thousands of seconds. The quality of their best solution is slightly better than ours in 60% of the instances tested. When comparing GRASP_B&B with the shifting bottleneck, ours is still faster, and it achieves better solutions, except for 3 of the comparable instances.

ACKNOWLEDGEMENT

S. Fernandes' work is suported by the the programm POCI2010 of the portuguese Fundação para a Ciência e Tecnologia. Helena R. Lourenço's work is supported by Ministerio de Educación y Ciencia, Spain, SEC2003-01991/ECO.

REFERENCES

[1] Adams, J., E. Balas and D. Zawack (1988). "The Shifting Bottleneck Procedure for Job Shop Scheduling." Management Science, vol. 34(3): pp. 391-401.

[2] Aiex, R. M., S. Binato and M. G. C. Resende (2003). "Parallel GRASP with path-relinking for job shop scheduling." Parallel Computing, vol. 29(4): pp. 393-430.

[3] Applegate, D. and W. Cook (1991). "A Computational Study of the Job-Shop Scheduling Problem." ORSA Journal on Computing, vol. 3(2): pp. 149-156.

[4] Balas, E. and A. Vazacopoulos (1998). "Guided Local Search with Shifting Bottleneck for Job Shop Scheduling." Management Science, vol. 44(2): pp. 262-275.

[5] Binato, S., W. J. Hery, D. M. Loewenstern and M. G. C. Resende (2002). "A GRASP for Job Shop Scheduling." In P. Hansen and C.C. Ribeiro, editors, Essays and surveys on metaheuristics. Kluwer Academic Publishers, 2001.

[6] Carlier, J. (1982). "The one-machine sequencing problem." European Journal of Operational Research, vol. 11: pp. 42-47.

[7] Caseau, Y. and F. Laburthe (1995), "Disjunctive scheduling with task intervals", Technical Report LIENS, 95-25, Ecole Normale Superieure Paris.

[8] Chen, S., S. Talukdar and N. Sadeh (1993). "Job-shop-scheduling by a team of asynchronous agentes", Proceedings of the IJCAI-93 Workshop on Knowledge-Based Production, Scheduling and Control. Chambery France.

[9] Danna, E., E. Rothberg and C. L. Pape (2005). "Exploring relaxation induced neighborhoods to improve MIP solutions." Mathematical Programming, Ser. A, vol. 102: pp. 71-90.

[10] Dell'Amico, M. and M. Trubian (1993). "Applying Tabu-Search to the Job-Shop Scheduling Problem."

[11] Denzinger, J. and T. Offermann (1999). "On Cooperation between Evolutionary Algorithms and other Search Paradigms", Proceedings of the 1999 Congress on Evolutionary Computational.

[12] Feo, T. and M. Resende (1995). "Greedy Randomized Adaptive Search Procedures." Journal of Global Optimization, vol. 6: pp. 109-133.

[13] Fernandes, S. (2002), "Técnicas heurísticas para o problema Job Shop Scheduling", Masters Thesis, Departamento de Estatística e Investigação Operacional, Faculdade de Ciências, Universidade de Lisboa.

[14] Fernandes, S. and H.R. Lourenço (2006), "Optimized Search methods", Working paper, Universitat Pompeu Fabra, Barcelona, Spain.

[15] Fisher, H. and G. L. Thompson (1963). Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson eds. Industrial Scheduling. pp. 225-251. Prentice Hall, Englewood Cliffs.

[16] 15 Garey, M. R. and D. S. Johnson (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco, Freeman.

[17] Jain, A. S. and S. Meeran (1999). "Deterministic job shop scheduling: Past, present and future." European Journal of Operational Research, vol. 133: pp. 390-434.

[18] Lawrence, S. (1984), "Resource Constrained Project Scheduling: an Experimental Investigation of Heuristic Scheduling techniques", Graduate School of Industrial Administration, Carnegie-Mellon University.

[19] Lourenço, H. R. (1995). "Job-shop scheduling: Computational study of local search and large-step optimization methods." European Journal of Operational Research, vol. 83: pp. 347-367.

[20] Lourenço, H. R. and M. Zwijnenburg (1996). Combining large-step optimization with tabu-search: Application to the job-shop scheduling problem. In I. H. Osman and J. P. Kelly eds. Meta-heuristics: Theory & Applications. Kluwer Academic Publishers.

[21] Nowicki, E. and C. Smutnicki (2002), "Some new tools to solve the job shop problem", Technical Report, 60/2002, Institute of Engineering Cybernetics, Wroclaw University of Technology.

[22] Nowicki, E. and C. Smutnicki (2005). "An Advanced Tabu Search Algorithm for the Job Shop Problem." Journal of Scheduling, vol. 8: pp. 145-159.

[23] Nowicki, E. and C. Smutniki (1996). "A Fast Taboo Search Algorithm for the Job Shop Problem." Management Science, vol. 42(6): pp. 797-813.

[24] Roy, B. and B. Sussman (1964), "Les probèms d'ordonnancement avec constraintes disjonctives", Note DS 9 bis, SEMA, Paris.

[25] Schaal, A., A. Fadil, H. M. Silti and P. Tolla (1999). "Meta heuristics diversification of generalized job shop scheduling based upon mathematical programming techniques", Proceedings of the Cp-ai-or'99.

[26] Schrage, L. (1970). "Solving resource-constrained network problems by implicit enumeration: Non pre-emptive case." Operations Research, vol. 18: pp. 263-278.

[27] Storer, R. H., S. D. Wu and R. Vaccari (1992). "New search spaces for sequencing problems with application to job shop scheduling." Management Science, vol. 38(10): pp. 1495-1509.

[28] Taillard, E. D. (1993). "Benchmarks for Basic Scheduling Problems." European Journal of Operational Research, vol. 64(2): pp. 278-285.

[29] Taillard, É. D. (1994). "Parallel Taboo Search Techniques for the Job Shop Scheduling Problem." ORSA Journal on Computing, vol. 6(2): pp. 108-117.

[30] Tamura, H., A. Hirahara, I. Hatono and M. Umano (1994). "An approximate solution method for combinatorial optimisation." Transactions of the Society of Instrument and Control Engineers, vol. 130: pp. 329-336.

[31] Yamada, T. and R. Nakano (1992). A genetic algorithm applicable to large-scale job-shop problems. In R. Manner and B. Manderick eds. Parallel Problem Solving from Nature 2. pp. 281-290. Elsevier Science.