A SIMPLE OPTIMISED SEARCH HEURISTIC FOR THE JOB-SHOP SCHEDULING PROBLEM

Susana Fernandes
Universidade do Algarve, Faro, Portugal.
E-mail: sfer@ualg.pt

Helena R. Lourenço
Univertitat Pompeu Fabra, Barcelona, Spain.
E-mail: helena.ramalhinho@upf.edu

Abstract: This paper presents a simple Optimised Search Heuristic for the Job Shop Scheduling problem that combines a GRASP heuristic with a branch-and-bound algorithm. The proposed method is compared with similar approaches and leads to better results in terms of solution quality and computing times.

Keywords: job-shop scheduling, hybrid metaheuristic, optimised search heuristics, GRASP, exact methods.
JEL Codes: C61, M11

1. 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 optimisation 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 methods applied to this particular 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 Optimised Search Heuristics (OSH) (Fernandes and Lourenço 2007).

This paper presents a simple OSH procedure for the job shop scheduling problem that combines a GRASP heuristic with a branch-and-bound algorithm.

In the next section, we introduce the job shop scheduling problem. In section 2, we present a short review of existent OSH methods applied to this problem and in

section 3 we describe in detail the OSH method developed. In section 5, we present the computational results along with comparisons to other similar procedures applied to the JSS problem. Section 6 concludes this paper and discusses some ideas for future research.

2. The job shop scheduling problem

The Job-Shop Scheduling Problem (JSSP) considers a set of jobs to be processed on 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). The order of the operations within the jobs and its correspondent machines are fixed a priori and independent from job to job. To solve the problem we need to find a sequence of operations on each machine respecting some constraints and optimising 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 dummy operations representing the start and end of all jobs, respectively. Let M be the set of machines, A the set of arcs between 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 decision variable representing the start time of operation i. The following mathematical formulation for the job shop scheduling problem is widely used by researchers:

($JSSP$)

$$
\begin{aligned}
s.t. \quad & \min t_{o+1} & & \\
& t_j - t_i \geq p_i & (i,j) \in A & \quad (1)\\
& t_i \geq 0 & i \in O & \quad (2)\\
& t_j - t_i \geq p_i \vee t_i - t_j \geq p_j & (i,j) \in E_k, k \in M & \quad (3)
\end{aligned}
$$

The constraints in (1) state the precedence of operations within jobs and also that no two operations of the same job can be processed simultaneously (because $p_i > 0$).

Expressions (3) are named “capacity constraints” and assure there are no overlaps of operations at the machines. A feasible solution for the problem is a schedule of operations respecting all these constraints.

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 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 $D_S = (O, A \cup S)$ where $S = \bigcup_k S_k$ 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). 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.

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.

3. Review of Optimised Search Heuristics

In the literature we can find a few works combining metaheuristics with exact algorithms applied to the job shop scheduling problem, designated as Optimized Search Heuristics (OSH) by Fernandes and Lourenço (2007). Different combinations of different procedures are present in the literature, and there are several applications of the OSH methods to different problems (see the web page of Fernandes and Lourenço (2007))[1].

[1] http://www.econ.upf.edu/~ramalhin/OSHwebpage/index.html

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 neighbourhood 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 a tabu search algorithm. The diversification of the search is achieved by applying a branch-and-bound method to solve a one-machine scheduling problem subproblem obtained from the incumbent solution.

In the work of Schaal Fadil Silti and Tolla (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 of 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 neighbourhood of the current best feasible solution. The local search consists in solving the restricted MIP problem defined by the neighbourhood.

4. Optimised search heuristic – GRASP_B&B

We developed a simple Optimised 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 means "Greedy Randomised Adaptive Search Procedure", (Feo and Resende 1995). It is an iterative process where each iteration consists of two steps: a randomised building step of a greedy nature and a local search step. At the building phase, a feasible solution is constructed joining one element at a time. Each element is evaluated by a greedy 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, if it has already more than one element, 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.

Next, we describe the OSH method GRASP_B&B developed to solve the Job-Shop Scheduling problem. The main spirit of this heuristic is combining a GRASP method with a branch-and-bound to efficiently solve the JSSP.

4.1 The Building step

In this section, we describe in detail the building step of the GRASP_B&B heuristic. We define the sequence of operations at each machine as the elements to join the solution, and the 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 (RCL), we find the optimal solution and optimal makespan, $f(x_k)$, for the one machine problems corresponding to all machines not yet scheduled. We identify the best $(\underline{f})$ and worst $(\overline{f})$ optimal makespans over all machines considered. 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 higher 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** $f(x_k) \geq \overline{f} - \alpha\left(\overline{f} - \underline{f}\right)$

(7) $RCL := RCL \cup \{k\}$

(8) **return** *RandomChoice*(*RCL*)

The building step requires a procedure to solve the one-machine scheduling problem. To solve this problem 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. This one machine scheduling problem considers that to each job j it is associated the following values that are obtained from the current solution: 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. Considering the job shop problem and its disjunctive graph representation, the release date of each operation i - (r_i) is obtained 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.

The one-machine branch-and-bound procedure implemented work as follows. 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 by preferring larger processing times. The computation of the lower bound is based on the critical path with more jobs of the solution found by the algorithm of Schrage (1970) and on a critical job, defined by some properties proved by Carlier (1982). 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, to reduce the computational times, 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). We designated this procedure as *Carlier_B&B(k)*, where k is the machine considered to be optimized and output the optimal one-machine schedule and the respective optimal value.

The way the one-machine branch-and-bound procedure is used within the building step is described next. 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. In terms of the mathematical formulation, this means choosing one of the inequalities of the disjunctive constraints (3) correspondent to the machine. We then update the makespan of the partial solution and the release dates and tails of unscheduled operations using the same procedure as the one used in the algorithm of Taillard (1994). We designate this procedure as *TAILLARD(x)* that computes the makespan of a partial solution x for the JSSP.

4.2 The Local Search step

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

We use a neighbourhood structure very similar to the NB neighbourhood of Dell'Amico and Trubian (1993) and the one of Balas and Vazacopoulos (1998). To describe the moves that define this neighbourhood 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 nomination of 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 $o+1$ ($L(v,o+1) \geq L(js(u),o+1)$).

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.

The neighbourhood considered in the GRASP_B&B is slightly different from the one considered in Dell'Amico and Trubian (1993) and Balas and Vazacopoulos (1998) since it considers partial solutions obtained at each iteration of the GRASP_B&B heuristic. Therefore the local search is applied to a partial solution where a subset of all machines is scheduled. This neighbourhood is designated by $N(x, M \setminus K)$, where x is a partial solution, M is the set of all machines and K is the set of machines not yet scheduled in the building phase. When inspecting the neighbourhood $N(x, M \setminus K)$, we stop whenever we find a neighbour with a best evaluation value than the makespan of x.

To evaluate the quality of a neighbour of a partial solution x, obtained 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 same procedure as the one used in the algorithm of Taillard (1994), *TAILLARD(x)*.

The local search phase consists in the two procedures described in pseudo-code below:

LocalSearch(*x,f(x), M \K*)

(1) $s := neighbour(x, f(x), M \backslash K)$
(2) **while** $s \neq x$
(3) $x := s$
(4) $s := neighbour(x, f(x), M \backslash K)$
(5) **return** s

Neighbour(*x,f(x),M\K*)

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

4.3 GRASP_B&B

In this section, we present the complete GRASP_B&B implemented, that considers the two phases previously described. 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 full GRASP_B&B method can be generally described by the pseudo-code as follows:

GRASP_B&B (*runs*)

(1) $M := \{1, \cdots, m\}$
(2) **for** $r = 1$ to *runs*
(3) $x := \{\ \}$
(4) $K := M$
(5) **while** $K \neq \{\ \}$

(6) **foreach** $k \in K$

(7) $x_k := CARLIER\_B\&B(k)$

(8) $k^* := SEMIGREEDY(K)$

(9) $x := x \cup x_{k^*}$

(10) $f(x) := TAILLARD(x)$

(11) $K := K \setminus \{k^*\}$

(12) **if** $|K| < |M| - 1$

(13) $x := LOCALSEARCH(x, M \setminus K)$

(14) **if** x^* not initialised **or** $f(x) < f^*$

(15) $x^* := x$

(16) $f^* := f(x)$

(17) **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 semigreedy (line (8)) and the algorithm of Taillard (line (10)) are executed m times.

5. Computational 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).

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 $(n * m)$, and the best lower and upper bound values (LB, UB) of the makespan. If the lower bound is omitted, the upper bound is optimal. We gathered the values of these bounds from the paper of Jain and Meeran (1999) and the papers of Nowicki and Smutnicki (1996; 2002 and 2005).

The algorithm has been run 100 times for each instance on a Pentium 4 CPU 2.80 GHz and coded in C. The tables also present some statistical values concerning the

makespan of the solutions found in the 100 runs, as well as the total time of all runs (ttime) and the time to the best solution found (btime), in seconds. The statistics of the makespan computed over the 100 runs are the minimum (min), the first quartile (Q1), the median (Q2), the third quartile (Q3) and the maximum (max). We chose this measures because they allow us to see how disperse are the values obtain by different runs, which give us an idea of the robustness of the algorithm. Within brackets, next to each value , is the correspondent percentage of relative error to the upperbound:

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

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 23 of the 152 instances used in this study.

name	**n*m**	**LB**	**UB**	**min**	**Q1**	**Q2**	**Q3**	**max**	**ttime (s)**	**btime(s)**
abz5	10*10		1234	1258 (1.94)	1312 (6.32)	1332 (7.94)	1358 (10.05)	1460 (18.31)	0.7650	0.0995
abz6	10*10		943	952 (0.95)	978.75 (3.79)	997 (5.73)	1012.5 (7.37)	1078 (14.32)	0.7660	0.3064
abz7	15*20		656	725 (10.52)	750.75 (14.44)	763 (16.31)	781 (19.05)	810 (23.48)	10.9070	3.4902
abz8	15*20	647	669	734 (9.72)	767 (14.65)	780 (16.59)	797.25 (19.17)	837 (25.11)	10.5160	1.8929
abz9	15*20	661	679	754 (11.05)	782.5 (15.24)	792 (16.64)	809 (19.15)	874 (28.72)	10.4690	1.3610

name	**n*m**	**LB**	**UB**	**min**	**Q1**	**Q2**	**Q3**	**max**	**ttime (s)**	**btime(s)**
ft06	6*6		55	55 (0.00)	59 (7.27)	59 (7.27)	61 (10.91)	66 (20.00)	0.1400	0.1274
ft10	10*10		930	970 (4.30)	1026.75 (10.40)	1046 (12.47)	1073.25 (15.40)	1144 (23.01)	1.0000	0.5800
ft20	20*5		1165	1283 (10.13)	1304 (11.93)	1318 (13.13)	1365 (17.17)	1409 (20.94)	0.4690	0.0094

name	**n*m**	**LB**	**UB**	**min**	**Q1**	**Q2**	**Q3**	**max**	**ttime(s)**	**btime(s)**
la01	10*5		666	666 (0.00)	666 (0.00)	666 (0.00)	666 (0.00)	694 (4.20)	0.1720	0.0017
la02	10*5		655	667 (1.83)	712 (8.70)	722 (10.23)	722 (10.23)	835 (27.48)	0.1560	0.0437
la03	10*5		597	605 (1.34)	605 (1.34)	640 (7.20)	701 (17.42)	701 (17.42)	0.2190	0.0066
la04	10*5		590	607	610	648	648	672	0.1710	0.0051

			(2.88)	(3.39)	(9.83)	(9.83)	(13.90)		
la05	10*5	593	593 (0.00)	593 (0.00)	593 (0.00)	593 (0.00)	593 (0.00)	0.1100	0.0011
la06	15*5	926	926 (0.00)	926 (0.00)	926 (0.00)	926 (0.00)	926 (0.00)	0.1710	0.0017
la07	15*5	890	890 (0.00)	890 (0.00)	890 (0.00)	890 (0.00)	936 (5.17)	0.2030	0.0020
la08	15*5	863	863 (0.00)	863 (0.00)	880 (1.97)	921 (6.72)	976 (13.09)	0.2970	0.0149
la09	15*5	951	951 (0.00)	951 (0.00)	951 (0.00)	951 (0.00)	953 (0.21)	0.2810	0.0028
la10	15*5	958	958 (0.00)	958 (0.00)	958 (0.00)	958 (0.00)	958 (0.00)	0.1410	0.0014
la11	20*5	1222	1222 (0.00)	1222 (0.00)	1222 (0.00)	1222 (0.00)	1284 (5.07)	0.2660	0.0027
la12	20*5	1039	1039 (0.00)	1039 (0.00)	1039 (0.00)	1039 (0.00)	1039 (0.00)	0.2650	0.0027
la13	20*5	1150	1150 (0.00)	1150 (0.00)	1150 (0.00)	1150 (0.00)	1223 (6.35)	0.3750	0.0038
la14	20*5	1292	1292 (0.00)	1292 (0.00)	1292 (0.00)	1292 (0.00)	1292 (0.00)	0.2180	0.0022
la15	20*5	1207	1207 (0.00)	1240 (2.73)	1295 (7.29)	1295 (7.29)	1295 (7.29)	0.9060	0.0453
la16	10*10	945	1012 (7.09)	1038.5 (9.89)	1049 (11.01)	1060 (12.17)	1099 (16.30)	0.7350	0.0221
la17	10*10	784	787 (0.38)	813.75 (3.79)	836.5 (6.70)	864.25 (10.24)	950 (21.17)	0.7660	0.0843
la18	10*10	848	854 (0.71)	879.25 (3.69)	895 (5.54)	924 (8.96)	1042 (22.88)	0.7500	0.3000
la19	10*10	842	861 (2.26)	893.75 (6.15)	917 (8.91)	940.5 (11.70)	1020 (21.14)	0.9690	0.4554
la20	10*10	902	920 (2.00)	960 (6.43)	976 (8.20)	1011.5 (12.14)	1080 (19.73)	0.8130	0.0813
la21	15*10	1046	1092 (4.40)	1154 (10.33)	1177.5 (12.57)	1210.25 (15.70)	1286 (22.94)	2.0460	0.1023
la22	15*10	927	955 (3.02)	999 (7.77)	1029.5 (11.06)	1063.5 (14.72)	1192 (28.59)	1.7970	0.9884
la23	15*10	1032	1049 (1.65)	1089.25 (5.55)	1111 (7.66)	1136 (10.08)	1268 (22.87)	1.8900	1.7388
la24	15*10	935	971 (3.85)	1016 (8.66)	1030 (10.16)	1054.25 (12.75)	1104 (18.07)	1.8440	0.6270
la25	15*10	977	1027 (5.12)	1082.75 (10.82)	1100 (12.59)	1122.25 (14.87)	1226 (25.49)	1.7960	0.5388
la26	20*10	1218	1265 (3.86)	1321.75 (8.52)	1355 (11.25)	1376 (12.97)	1485 (21.92)	3.3750	3.0375
la27	20*10	1235	1308 (5.91)	1375 (11.34)	1399 (13.28)	1431.25 (15.89)	1538 (24.53)	3.5620	0.1781
la28	20*10	1216	1301 (6.99)	1360.75 (11.90)	1391 (14.39)	1413.25 (16.22)	1533 (26.07)	3.0000	0.1500

la29	20*10		1152	1248 (8.33)	1312.75 (13.95)	1339 (16.23)	1379 (19.70)	1466 (27.26)	3.2960	0.8570
la30	20*10		1355	1382 (1.99)	1432.75 (5.74)	1452.5 (7.20)	1477 (9.00)	1548 (14.24)	3.3280	0.8653
la31	30*10		1784	1784 (0.00)	1806.75 (1.28)	1829.5 (2.55)	1866.25 (4.61)	2006 (12.44)	7.0160	0.0702
la32	30*10		1850	1850 (0.00)	1868.75 (1.01)	1906 (3.03)	1931 (4.38)	2024 (9.41)	6.2350	0.5612
la33	30*10		1719	1719 (0.00)	1729.75 (0.63)	1756.5 (2.18)	1797 (4.54)	1872 (8.90)	7.9060	1.2650
la34	30*10		1721	1721 (0.00)	1787 (3.83)	1812 (5.29)	1845.25 (7.22)	2025 (17.66)	8.2810	3.8093
la35	30*10		1888	1888 (0.00)	1901 (0.69)	1923 (1.85)	1978.25 (4.78)	2232 (18.22)	5.6880	0.2844
la36	15*15		1268	1325 (4.50)	1375.75 (8.50)	1395.5 (10.06)	1423.25 (12.24)	1521 (19.95)	4.2650	0.0853
la37	15*15		1397	1479 (5.87)	1538.75 (10.15)	1565.5 (12.06)	1597.25 (14.33)	1642 (17.54)	4.7970	4.0295
la38	15*15		1196	1274 (6.52)	1354.75 (13.27)	1381.5 (15.51)	1397.75 (16.87)	1471 (22.99)	5.1090	0.7153
la39	15*15		1233	1309 (6.16)	1352.75 (9.71)	1374 (11.44)	1404.25 (13.89)	1468 (19.06)	4.4530	2.9835
la40	15*15		1222	1291 (5.65)	1347 (10.23)	1369 (12.03)	1398.5 (14.44)	1451 (18.74)	5.3910	3.5581

name	**n*m**	**LB**	**UB**	**min**	**Q1**	**Q2**	**Q3**	**max**	**ttime(s)**	**btime(s)**
orb01	10*10		1059	1145 (8.12)	1181.75 (11.59)	1198 (13.13)	1219.25 (15.13)	1335 (26.06)	0.9850	0.0296
orb02	10*10		888	918 (3.38)	959.75 (8.08)	983 (10.70)	1013.25 (14.10)	1085 (22.18)	0.9530	0.0953
orb03	10*10		1005	1098 (9.25)	1135.5 (12.99)	1155.5 (14.98)	1184.25 (17.84)	1289 (28.26)	1.0150	0.3350
orb04	10*10		1005	1066 (6.07)	1120 (11.44)	1144.5 (13.88)	1183 (17.71)	1255 (24.88)	1.1250	0.8213
orb05	10*10		887	911 (2.71)	966.75 (8.99)	1001 (12.85)	1014.25 (14.35)	1117 (25.93)	0.8750	0.1050
orb06	10*10		1010	1050 (3.96)	1108 (9.70)	1134.5 (12.33)	1172 (16.04)	1282 (26.93)	1.0460	0.4812
orb07	10*10		397	414 (4.28)	436.5 (9.95)	448 (12.85)	455 (14.61)	503 (26.70)	1.0630	0.2764
orb08	10*10		899	945 (5.12)	975 (8.45)	999 (11.12)	1032.75 (14.88)	1125 (25.14)	1.0310	0.3093
orb09	10*10		934	978 (4.71)	1003.75 (7.47)	1021 (9.31)	1053.75 (12.82)	1177 (26.02)	0.9060	0.2809
orb10	10*10		944	991 (4.98)	1024.75 (8.55)	1040 (10.17)	1073 (13.67)	1232 (30.51)	0.8430	0.2276

name	**n*m**	**LB**	**UB**	**min**	**Q1**	**Q2**	**Q3**	**max**	**ttime (s)**	**btime (s)**

swv01	20*10		1407	1605 (14.07)	1688 (19.97)	1762 (25.23)	1806.75 (28.41)	1900 (35.04)	3.6720	3.6720
swv02	20*10		1475	1601 (8.54)	1696 (14.98)	1729 (17.22)	1776.5 (20.44)	1940 (31.53)	3.2650	0.2939
swv03	20*10	1369	1398	1582 (13.16)	1666.75 (19.22)	1704.5 (21.92)	1738.5 (24.36)	1964 (40.49)	3.4850	1.4986
swv04	20*10	1450	1483	1655 (11.60)	1737.5 (17.16)	1772.5 (19.52)	1816.25 (22.47)	1949 (31.42)	4.0000	2.7200
swv05	20*10		1424	1587 (11.45)	1660.75 (16.63)	1690 (18.68)	1718.25 (20.66)	1829 (28.44)	3.6720	3.5986
swv06	20*15	1591	1678	1895 (12.93)	1975 (17.70)	2012.5 (19.93)	2064.25 (23.02)	2240 (33.49)	10.7810	8.0858
swv07	20*15	1446	1620	1833 (13.15)	1881.75 (16.16)	1921 (18.58)	1953.75 (20.60)	2076 (28.15)	11.5470	2.3094
swv08	20*15	1640	1763	2001 (13.50)	2103.25 (19.30)	2150 (21.95)	2190 (24.22)	2318 (31.48)	11.0310	9.0454
swv09	20*15	1604	1663	1877 (12.87)	1984.75 (19.35)	2017.5 (21.32)	2088 (25.56)	2197 (32.11)	11.3900	10.0232
swv10	20*15	1631	1767	1978 (11.94)	2053.5 (16.21)	2102 (18.96)	2145 (21.39)	2288 (29.49)	10.0630	4.4277
swv11	50*10	2983	2991	3366 (12.54)	3454.25 (15.49)	3498.5 (16.97)	3574 (19.49)	4047 (35.31)	62.3590	8.7303
swv12	50*10	2972	3003	3422 (13.95)	3520.75 (17.24)	3569 (18.85)	3621 (20.58)	4196 (39.73)	141.9220	5.6769
swv13	50*10		3104	3527 (13.63)	3601.25 (16.02)	3654 (17.72)	3698.25 (19.14)	4143 (33.47)	54.9840	20.3441
swv14	50*10		2968	3295 (11.02)	3362.25 (13.28)	3402.5 (14.64)	3469.5 (16.90)	4052 (36.52)	180.8440	159.1427
swv15	50*10	2885	2904	3329 (14.63)	3458.5 (19.09)	3565 (22.76)	3634.25 (25.15)	3994 (37.53)	113.1720	73.5618
swv16	50*10		2924	**2924** (0.00)	**2924** (0.00)	**2924** (0.00)	**2924** (0.00)	2962 (1.30)	9.6720	0.0967
swv17	50*10		2794	**2794** (0.00)	**2794** (0.00)	2798 (0.14)	2828 (1.22)	2949 (5.55)	16.9690	0.6788
swv18	50*10		2852	**2852** (0.00)	**2852** (0.00)	**2852** (0.00)	2879 (0.95)	2985 (4.66)	15.6090	0.1561
swv19	50*10		2843	**2843** (0.00)	2864 (0.74)	2904 (2.15)	2972.5 (4.56)	3168 (11.43)	30.2650	2.1186
swv20	50*10		2823	**2823** (0.00)	**2823** (0.00)	2846.5 (0.83)	2894.25 (2.52)	3045 (7.86)	17.3900	0.8695

name	**n*m**	**LB**	**UB**	**min**	**Q1**	**Q2**	**Q3**	**max**	**ttime (s)**	**btime (s)**
yn1	20*20	826	888	955 (7.55)	996.75 (12.25)	1010.5 (13.80)	1031.25 (16.13)	1084 (22.07)	23.4530	4.2215
yn2	20*20	861	909	987 (8.58)	1035.75 (13.94)	1047 (15.18)	1060 (16.61)	1133 (24.64)	25.3750	12.4338
yn3	20*20	827	893	996 (11.53)	1029.75 (15.31)	1049 (17.47)	1068.5 (19.65)	1111 (24.41)	25.3430	11.9112
yn4	20*20	918	968	1060	1117.75	1132	1158	1209	23.8900	20.0676

				(9.50)	(15.47)	(16.94)	(19.63)	(24.90)		

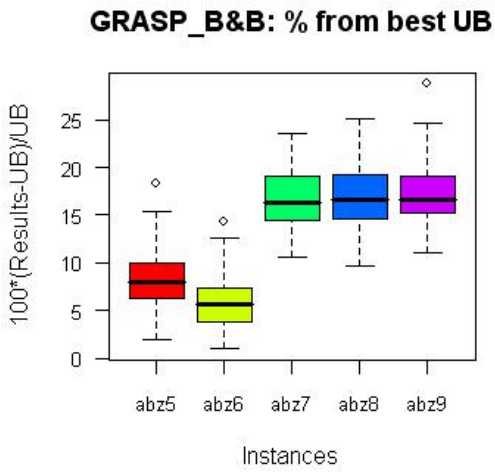
name	N*m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
ta01	15*15		1231	1332 (8.20)	1387 (12.67)	1413 (14.78)	1438.25 (16.84)	1556 (26.40)	5.5630	0.2782
ta02	15*15		1244	1313 (5.55)	1368.75 (10.03)	1394.5 (12.10)	1426.75 (14.69)	1499 (20.50)	5.5150	4.9635
ta03	15*15		1218	1278 (4.93)	1346.75 (10.57)	1370 (12.48)	1403.25 (15.21)	1488 (22.17)	5.4060	4.1086
ta04	15*15		1175	1249 (6.30)	1309 (11.40)	1330.5 (13.23)	1360.25 (15.77)	1518 (29.19)	6.1560	3.4474
ta05	15*15		1224	1310 (7.03)	1369 (11.85)	1393.5 (13.85)	1432 (16.99)	1579 (29.00)	5.8120	0.8718
ta06	15*15		1238	1308 (5.65)	1362.75 (10.08)	1396 (12.76)	1422.5 (14.90)	1535 (23.99)	5.9370	0.7718
ta07	15*15		1227	1299 (5.87)	1342 (9.37)	1364 (11.17)	1390.25 (13.30)	1549 (26.24)	5.1400	2.3130
ta08	15*15		1217	1306 (7.31)	1371 (12.65)	1389.5 (14.17)	1414.25 (16.21)	1523 (25.14)	5.9530	2.7979
ta09	15*15		1274	1395 (9.50)	1438 (12.87)	1465 (14.99)	1491 (17.03)	1614 (26.69)	6.1100	1.5886
ta10	15*15		1241	1332 (7.33)	1387 (11.76)	1413 (13.86)	1438.25 (15.89)	1556 (25.38)	5.5150	0.2758
ta11	20*15	1323	1361	1497 (9.99)	1571 (15.43)	1597.5 (17.38)	1626.25 (19.49)	1727 (26.89)	11.1100	7.9992
ta12	20*15	1351	1367	1511 (10.53)	1576.75 (15.34)	1590.5 (16.35)	1623.25 (18.75)	1709 (25.02)	11.7650	3.4119
ta13	20*15	1282	1342	1498 (11.62)	1559.5 (16.21)	1581.5 (17.85)	1618.25 (20.58)	1728 (28.76)	10.6880	5.0234
ta14	20*15		1345	1439 (6.99)	1496.75 (11.28)	1527.5 (13.57)	1569 (16.65)	1692 (25.80)	11.5940	2.3188
ta15	20*15	1304	1340	1511 (12.76)	1576.25 (17.63)	1602 (19.55)	1639 (22.31)	1732 (29.25)	12.4380	4.3533
ta16	20*15	1302	1360	1486 (9.26)	1551.75 (14.10)	1571.5 (15.55)	1609.25 (18.33)	1677 (23.31)	11.2030	5.9376
ta17	20*15		1462	1600 (9.44)	1661 (13.61)	1693.5 (15.83)	1713 (17.17)	1911 (30.71)	9.3910	7.7006
ta18	20*15	1369	1396	1543 (10.53)	1623 (16.26)	1652 (18.34)	1677.25 (20.15)	1782 (27.65)	12.2030	7.5659
ta19	20*15	1297	1335	1463 (9.59)	1542 (15.51)	1574 (17.90)	1616 (21.05)	1740 (30.34)	11.5310	7.2645
ta20	20*15	1318	1351	1498 (10.88)	1549 (14.66)	1580 (16.95)	1617 (19.69)	1686 (24.80)	11.5620	6.2435
ta21	20*20	1539	1644	1810 (10.10)	1894.5 (15.24)	1936 (17.76)	1970.25 (19.84)	2144 (30.41)	20.9690	11.1136
ta22	20*20	1511	1600	1792 (12.00)	1832.75 (14.55)	1865 (16.56)	1903 (18.94)	1989 (24.31)	22.4840	6.0707
ta23	20*20	1472	1557	1708 (9.70)	1768.75 (13.60)	1801 (15.67)	1839.25 (18.13)	1947 (25.05)	22.0780	16.1169

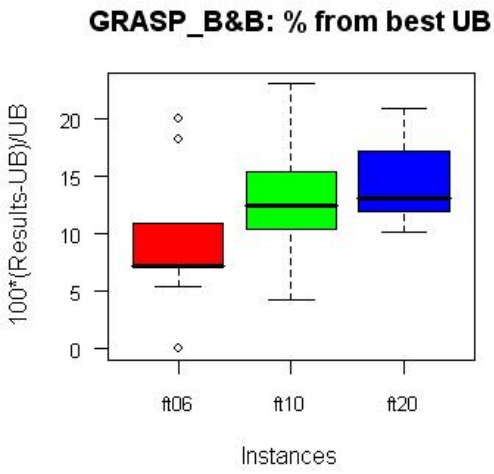
ta24	20*20	1602	1647	1778 (7.95)	1864.75 (13.22)	1894.5 (15.03)	1925.25 (16.89)	2014 (22.28)	19.1870	17.0764
ta25	20*20	1504	1595	1746 (9.47)	1830.75 (14.78)	1876 (17.62)	1913.5 (19.97)	1992 (24.89)	20.4060	16.1207
ta26	20*20	1539	1645	1768 (7.48)	1863.75 (13.30)	1907 (15.93)	1950.25 (18.56)	2027 (23.22)	17.8440	2.6766
ta27	20*20	1616	1680	1839 (9.46)	1923.75 (14.51)	1954 (16.31)	1988.25 (18.35)	2149 (27.92)	19.8440	17.8596
ta28	20*20	1591	1614	1755 (8.74)	1837.5 (13.85)	1871 (15.92)	1908.75 (18.26)	2016 (24.91)	22.3130	19.1892
ta29	20*20	1514	1625	1717 (5.66)	1835.75 (12.97)	1864 (14.71)	1898.25 (16.82)	2012 (23.82)	20.1570	6.2487
ta30	20*20	1473	1584	1737 (9.66)	1800.75 (13.68)	1827.5 (15.37)	1852.5 (16.95)	1960 (23.74)	17.5470	6.4924
ta31	30*15		1764	1976 (12.02)	2059.75 (16.77)	2099.5 (19.02)	2135 (21.03)	2322 (31.63)	30.6090	14.3862
ta32	30*15	1774	1796	2029 (12.97)	2132.5 (18.74)	2165.5 (20.57)	2205 (22.77)	2356 (31.18)	30.6090	23.8750
ta33	30*15	1778	1793	2070 (15.45)	2171.25 (21.10)	2204 (22.92)	2265 (26.32)	2336 (30.28)	31.9220	18.1955
ta34	30*15	1828	1829	2024 (10.66)	2114.25 (15.60)	2156 (17.88)	2186 (19.52)	2287 (25.04)	33.6720	32.3251
ta35	30*15		2007	2093 (4.29)	2174.25 (8.33)	2208 (10.01)	2250.25 (12.12)	2454 (22.27)	35.2970	13.4129
ta36	30*15		1819	2040 (12.15)	2124.5 (16.79)	2153.5 (18.39)	2204.25 (21.18)	2307 (26.83)	30.5310	3.9690
ta37	30*15	1771	1778	1967 (10.63)	2099.75 (18.10)	2136 (20.13)	2184.5 (22.86)	2408 (35.43)	29.0310	15.3864
ta38	30*15		1673	1913 (14.35)	1976.75 (18.16)	2003.5 (19.75)	2046.25 (22.31)	2188 (30.78)	34.0000	12.5800
ta39	30*15		1795	1966 (9.53)	2069.25 (15.28)	2107.5 (17.41)	2144 (19.44)	2321 (29.30)	31.3750	13.4913
ta40	30*15	1631	1674	1931 (15.35)	2012 (20.19)	2047.5 (22.31)	2077.25 (24.09)	2218 (32.50)	33.9850	0.3399
ta41	30*20	1859	2018	2348 (16.35)	2413.75 (19.61)	2458 (21.80)	2494 (23.59)	2638 (30.72)	61.1100	0.6111
ta42	30*20	1867	1956	2206 (12.78)	2301.75 (17.68)	2341 (19.68)	2374.25 (21.38)	2513 (28.48)	62.3280	49.2391
ta43	30*20	1809	1859	2155 (15.92)	2228.5 (19.88)	2254 (21.25)	2294.5 (23.43)	2395 (28.83)	72.7040	50.8928
ta44	30*20	1927	1984	2300 (15.93)	2382.5 (20.09)	2418 (21.88)	2466 (24.29)	2577 (29.89)	64.7970	11.6635
ta45	30*20	1997	2000	2295 (14.75)	2358 (17.90)	2380 (19.00)	2410.25 (20.51)	2581 (29.05)	70.8280	34.7057
ta46	30*20	1940	2021	2314 (14.50)	2399.5 (18.73)	2438 (20.63)	2481 (22.76)	2660 (31.62)	64.1090	25.6436
ta47	30*20	1789	1903	2151 (13.03)	2260.5 (18.79)	2299.5 (20.84)	2345.75 (23.27)	2443 (28.38)	63.9850	63.3452
ta48	30*20	1912	1952	2222	2325.5	2360	2407.5	2565	63.2190	60.0581

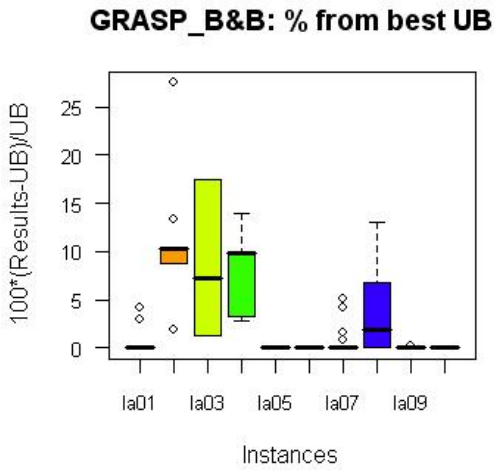
				(13.83)	(19.13)	(20.90)	(23.34)	(31.40)		
ta49	30*20	1915	1968	2250 (14.33)	2349.5 (19.39)	2390 (21.44)	2425 (23.22)	2560 (30.08)	65.2650	11.7477
ta50	30*20	1807	1928	2264 (17.43)	2347.5 (21.76)	2387 (23.81)	2431 (26.09)	2633 (36.57)	63.2970	13.2924
ta51	50*15		2760	3001 (8.73)	3149 (14.09)	3227 (16.92)	3294 (19.35)	3587 (29.96)	139.8750	65.7413
ta52	50*15		2756	2940 (6.68)	3136 (13.79)	3216 (16.69)	3276.5 (18.89)	3590 (30.26)	128.3590	119.3739
ta53	50*15		2717	2895 (6.55)	2998 (10.34)	3031.5 (11.58)	3072.5 (13.08)	3219 (18.48)	116.9370	87.7028
ta54	50*15		2839	2914 (2.64)	3024.75 (6.54)	3092.5 (8.93)	3133.75 (10.38)	3280 (15.53)	112.5470	15.7566
ta55	50*15		2679	2988 (11.53)	3133 (16.95)	3178 (18.63)	3247.5 (21.22)	3440 (28.41)	144.6570	56.4162
ta56	50*15		2781	2966 (6.65)	3122.75 (12.29)	3167 (13.88)	3230.25 (16.15)	3437 (23.59)	131.3750	3.9413
ta57	50*15		2943	3101 (5.37)	3213.5 (9.19)	3256.5 (10.65)	3320.5 (12.83)	3485 (18.42)	106.4220	27.6697
ta58	50*15		2885	3103 (7.56)	3214.5 (11.42)	3273 (13.45)	3326.25 (15.29)	3438 (19.17)	146.9060	104.3033
ta59	50*15		2655	2940 (10.73)	3038.25 (14.44)	3090 (16.38)	3139.5 (18.25)	3294 (24.07)	124.2970	119.3251
ta60	50*15		2723	2921 (7.27)	3048.5 (11.95)	3104 (13.99)	3160 (16.05)	3339 (22.62)	121.7970	86.4759
ta61	50*20		2868	3258 (13.60)	3369.5 (17.49)	3424 (19.39)	3484 (21.48)	3649 (27.23)	285.5000	216.9800
ta62	50*20	2869	2872	3306 (15.11)	3444 (19.92)	3493.5 (21.64)	3544 (23.40)	3730 (29.87)	311.9850	293.2659
ta63	50*20		2755	3130 (13.61)	3218.75 (16.83)	3273 (18.80)	3324.25 (20.66)	3487 (26.57)	315.1250	302.5200
ta64	50*20		2702	3008 (11.32)	3180.25 (17.70)	3230 (19.54)	3287.75 (21.68)	3431 (26.98)	289.8440	153.6173
ta65	50*20		2725	3104 (13.91)	3242.5 (18.99)	3286 (20.59)	3339 (22.53)	3569 (30.97)	323.2340	42.0204
ta66	50*20		2845	3198 (12.41)	3320 (16.70)	3361 (18.14)	3415.5 (20.05)	3585 (26.01)	317.1570	63.4314
ta67	50*20		2825	3209 (13.59)	3339 (18.19)	3389.5 (19.98)	3435.25 (21.60)	3578 (26.65)	273.3440	46.4685
ta68	50*20		2784	3133 (12.54)	3235.75 (16.23)	3283 (17.92)	3337.5 (19.88)	3542 (27.23)	268.7810	223.0882
ta69	50*20		3071	3366 (9.61)	3447.5 (12.26)	3517 (14.52)	3576 (16.44)	3739 (21.75)	259.1720	33.6924
ta70	50*20		2995	3449 (15.16)	3528.75 (17.82)	3582 (19.60)	3633.25 (21.31)	3815 (27.38)	279.5150	58.6982

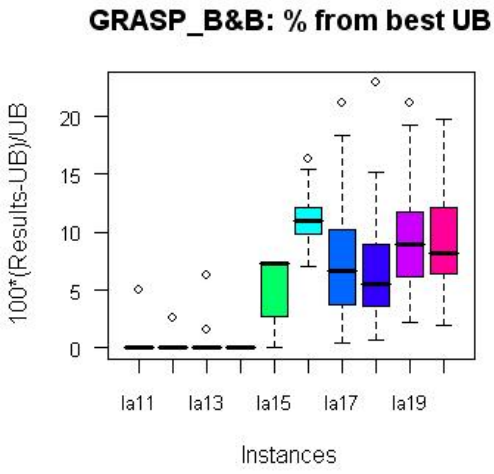
The information of these tables can be visualised using boxplots. They show that the quality achieved is more dependent on the ratio n/m than on the absolute

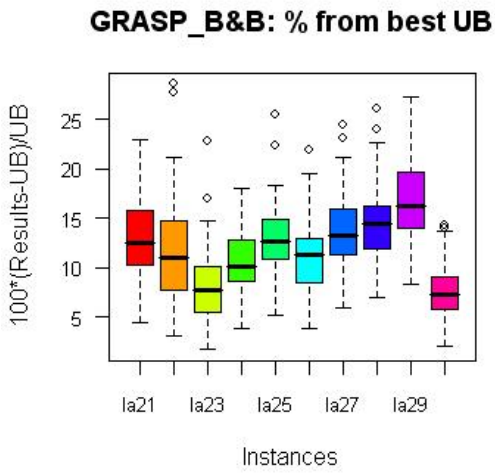
numbers of jobs and machines. There is no big dispersion of the solution values achieved by the algorithm in the 100 runs executed, so we say the algorithm is steady. The number of times the algorithm achieves the best values reported is high enough, so these values are not considered outliers of the distribution of the results. On the other end, the worse values occur very seldom and are outliers for the majority of the instances.

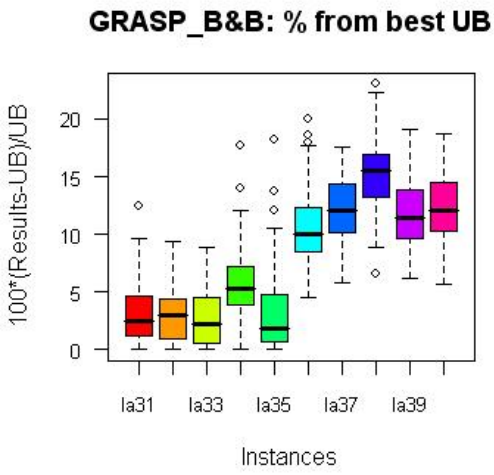


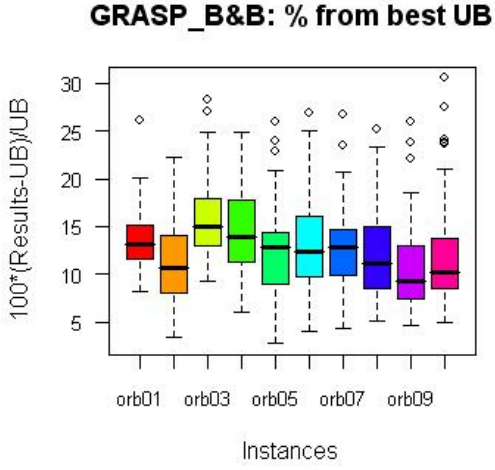


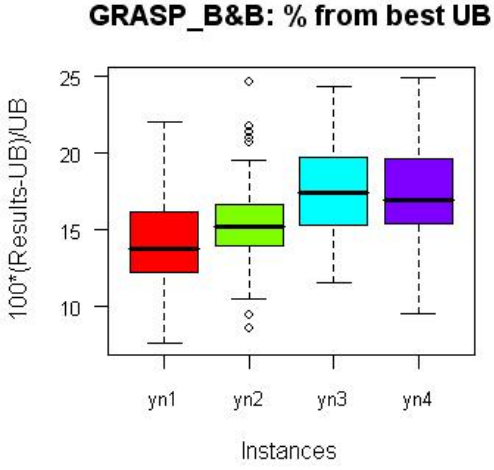


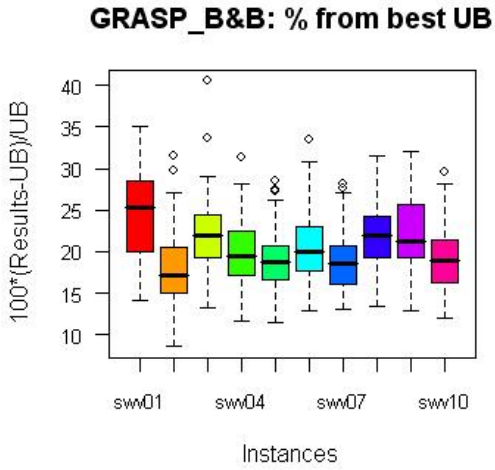


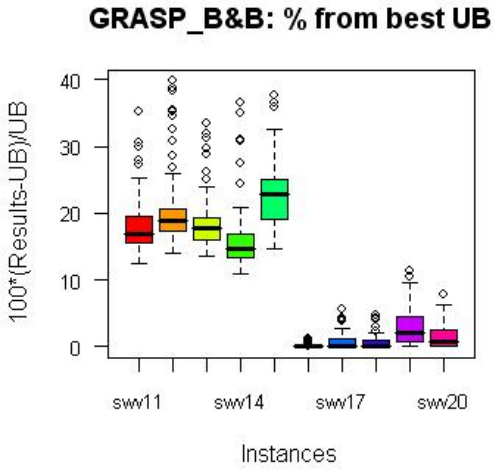

GRASP_B&B: % from best UB
100*(Results-UB)/UB
5
10
15
20
25
la21
la23
la25
la27
la29
Instances

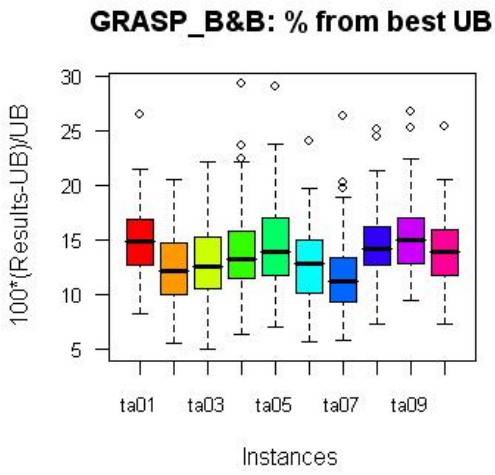

GRASP_B&B: % from best UB
100*(Results-UB)/UB
0
5
10
15
20
la31
la33
la35
la37
la39
Instances

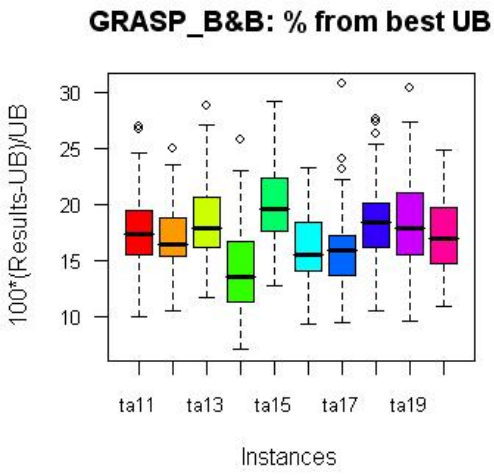

GRASP_B&B: % from best UB
100*(Results-UB)/UB
5
10
15
20
25
30
orb01
orb03
orb05
orb07
orb09
Instances

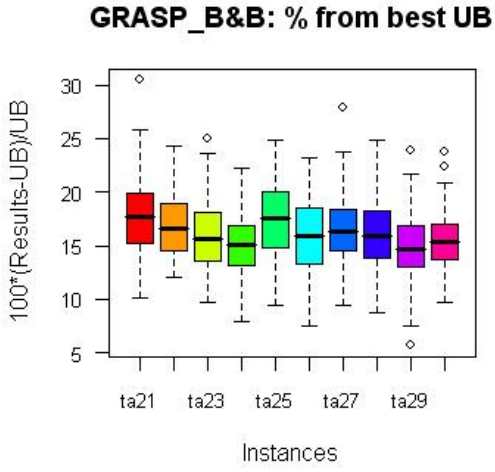

GRASP_B&B: % from best UB
100*(Results-UB)/UB
10
15
20
25
yn1
yn2
yn3
yn4
Instances

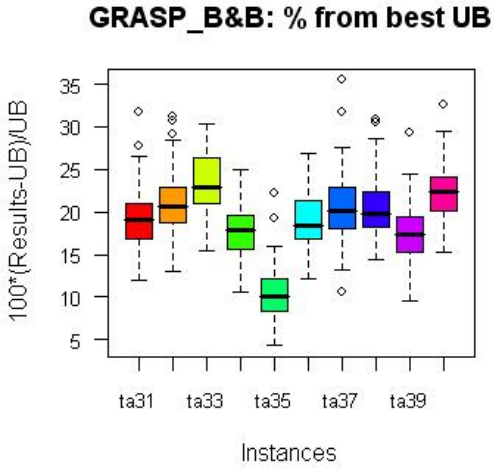

GRASP_B&B: % from best UB
100*(Results-UB)/UB
10
15
20
25
30
35
40
swv01
swv04
swv07
swv10
Instances

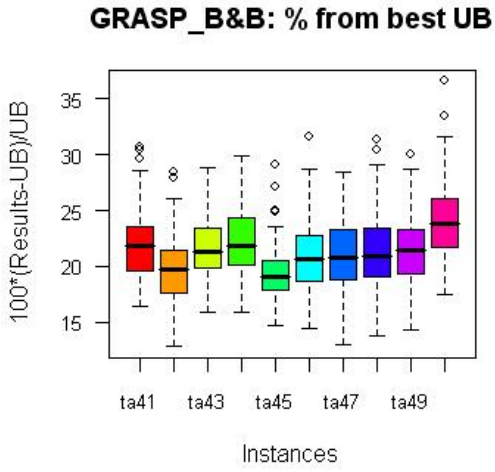

GRASP_B&B: % from best UB
100*(Results-UB)/UB
0
10
20
30
40
swv11
swv14
swv17
swv20
Instances

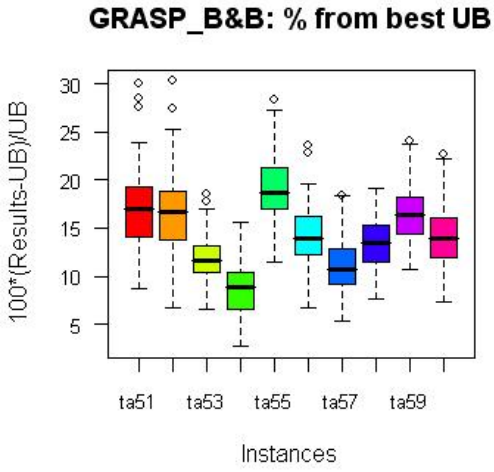

GRASP_B&B: % from best UB
100*(Results-UB)/UB
30
25
20
15
10
5
ta01
ta03
ta05
ta07
ta09
Instances

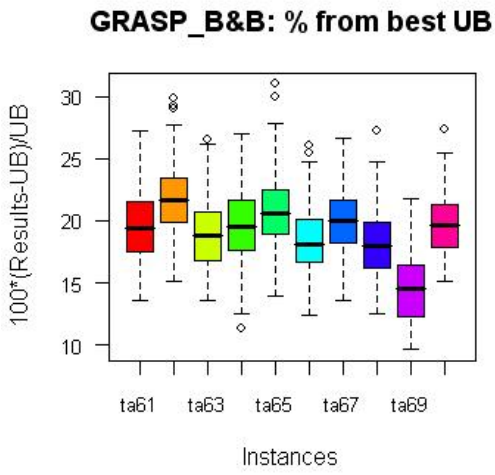

GRASP_B&B: % from best UB
100*(Results-UB)/UB
30
25
20
15
10
ta11
ta13
ta15
ta17
ta19
Instances


GRASP_B&B: % from best UB
100*(Results-UB)/UB
30
25
20
15
10
5
ta21
ta23
ta25
ta27
ta29
Instances


GRASP_B&B: % from best UB
100*(Results-UB)/UB
35
30
25
20
15
10
5
ta31
ta33
ta35
ta37
ta39
Instances


GRASP_B&B: % from best UB
100*(Results-UB)/UB
35
30
25
20
15
ta41
ta43
ta45
ta47
ta49
Instances


GRASP_B&B: % from best UB
100*(Results-UB)/UB
30
25
20
15
10
5
ta51
ta53
ta55
ta57
ta59
Instances



5.1. Comparison to other procedures

GRASP_B&B OSH heuristic is a very simple GRASP algorithm with a construction phase very similar to the one of the shifting bottleneck. Therefore, we show comparative results to two other very similar methods: a simple GRASP heuristic of Binato et al (2001) and the Shifting Bottleneck heuristic by Adams et al (1988).

5.1.1. Comparison to GRASP of Binato et al (2001)

The GRASP heuristic by Binato et al (2001) has a different building step in the construction phase, which consists in scheduling one operation at each step. 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. For GRASP_B&B we present the total time of all runs (ttime), in seconds. 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 26 of the 58 instances whose results where compared.

name	GRASP_B&B	ttime (s)	GRASP	time (s)
abz5	1258	0.7650	1238	6030
abz6	952	0.7660	947	62310
abz7	725	10.9070	667	349740
abz8	734	10.5160	729	365820

abz9	**754**	10.4690	758	343710

name	GRASP_B&B	ttime (s)	GRASP	time (s)
ft06	**55**	0.1400	55	70
ft10	970	1.0000	938	261290
ft20	1283	0.4690	1169	387430

name	GRASP_B&B	ttime (s)	GRASP	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

name	GRASP_B&B	ttime (s)	GRASP	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

5.1.2. Comparison to the Shifting Bottleneck (Adams et al. 1988)

The main difference of the Shifting Bottleneck procedure (Adams et al. 1988) and GRASP_B&B is the random selection of the machine to be scheduled. In the Shifting Bottleneck the machine to be scheduled is always the bottleneck machine. The comparison between the shifting bottleneck procedure (Adams et al. 1988) and the GRASP_B&B is also presented next. Comparing the computation times of both procedures, the GRASP_B&B 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 its 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.

name	GRASP_B&B	ttime (s)	Shifting Bottleneck	time (s)
abz5	**1258**	0.7650	1306	5.7
abz6	**952**	0.7660	962	12.67
abz7	**725**	10.9070	730	118.87
abz8	**734**	10.5160	774	125.02
abz9	754	10.4690	751	94.32

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

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	95º1	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

6. Conclusions

In this work we present a very simple Optimized Search Heuristic, the GRASP_B&B to solve the Job-Shop Scheduling problem. This method is intended to be a starting point for a more elaborated metaheuristic, since it obtains reasonable solutions in very short running times. The main idea behind the GRASP_B&B heuristic is to insert in each iteration of the building phase of the GRASP method the complete solution of one-machine scheduling problems solved by a branch-an-bound method, instead of insert one sequence of two individual operations as it is usual in other GRASP methods for this problem.

We have compared it with other similar methods also used as an initialization phase within more complex algorithms; namely a GRASP of Binato et. al (2001), which is the base for a GRASP with path-relinking procedure of Aiex et. al (2003), and the Shifting Bottleneck procedure of Adams et. al (1988), incorporated in the successful guided local search of Balas and Vazacopoulos (1991). The comparison to the GRASP of Binato et al (2001) shows that the GRASP_B&B is much faster than theirs. 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, the first one is still faster, and it achieves better solutions, except for 3 of the comparable

instances. Therefore we can conclude, that the GRASP_B&B is a good method to use as the initialization phase of more elaborated and complex methods to solve the job-shop scheduling problem. As future research, we are working on this elaborated method also using OSH ideas, i.e. combing heuristic and exact methods procedures.

Acknowledgement

Susana 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 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.

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

4 Binato, S., W. J. Hery, D. M. Loewenstern and M. G. C. Resende **(2001)**. "A GRASP for Job Shop Scheduling." In C.C. Ribeiro and P. Hansen, editors, Essays and surveys on metaheuristics, pp. 59-79. Kluwer Academic Publishers.

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

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

7 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.

8 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.

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

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

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

12 Fernandes, S. and H.R. Lourenço (2007), "Optimized Search Heuristics", Universitat Pompeu Fabra, Barcelona, Spain.
(http://www.econ.upf.edu/~ramalhin/OSHwebpage/index.html)

13 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.

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

15 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.

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

17 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.

18 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.

19 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.

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

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

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

23 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.

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

25 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.

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

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

28 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.

29 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.