A GRASP and branch-and-bound metaheuristic for the job-shop scheduling

A GRASP and Branch-and-Bound Metaheuristic for the Job-Shop Scheduling

Susana Fernandes and Helena R. Lourenço

Universidade do Algarve, Faro, Portugal
sfer@ualg.pt
Universitat Pompeu Fabra, Barcelona, Spain
helena.ramalhinho@upf.edu

Abstract. This paper presents a simple algorithm for the job shop scheduling problem that combines the local search heuristic GRASP 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 computing times.

1 Introduction

The job-shop scheduling problem has been known to the operations research community since the early 50's [16]. It is considered a particularly hard combinatorial optimization problem of the NP-hard class [15] 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 [16] 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. Fernandes and Lourenço [13] designated these methods by Optimized Search Heuristics (OSH).

In this paper we present 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.

2 The Job-Shop Scheduling Problem

The job-shop scheduling problem 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 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 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 variable representing the start time of operation i. The following mathematical formulation for the job shop scheduling problem is widely used:

The constraints in (1) state the precedences 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 on the machines.

A common representation for the job-shop problem is the disjunctive graph $G = (O, A, E)$ [22]; 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. 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. 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 or makespan.

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

Fig. 1. Mathematical formulation for the Job-Shop Problem

3 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, designated as Optimized Search Heuristics (OSH) by Fernandes and Lourenço [13]. Different combinations of different procedures are present in the literature, and there are several applications of the OSH methods to different problems.

Chen et al. [8] and Denzinger and Offermann [11] 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. [27] 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 [1], [3], [7] and [4] use an exact algorithm to solve a sub problem within a local search heuristic for the job shop scheduling. Caseau and Laburthe [7] build a local search where the neighborhood structure is defined by a subproblem that is solved exactly using constraint programming.

Applegate and Cook [3] 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 [1], 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 [4] work with the shifting bottleneck heuristic and design a guided local search, over a tree search structure, that reconstructs partially destroyed solutions.

Lourenço [18] and Lourenço and Zwijnenburg [19] use branch-and-bound algorithms to strategically guide an iterated local search and a tabu search algorithm. The diversification of the search is achieved applying a branch-and-bound method to solve a one-machine scheduling problem subproblem obtained from the incumbent solution.

The interesting work done by Danna, Rothberg and Le Pape [9] "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.

4 GRASP and Branch-and-Bound

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

GRASP [12] 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 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). 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 and the local optimum updates the current solution. The all process is repeated until the solution is complete.

4.1 Building Step

We define the sequence of operations at each machine as the elements to join the solution, and the makespan as the greedy function to evaluate them. In order to build a restricted candidate list of this elements (RCL), we solve exactly all the one machine problems and identify the best $(\underline{f})$ and worst $(\overline{f})$ 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 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.

To solve the one machine scheduling problems we use the branch-and-bound algorithm of Carlier [6]. The objective function of the algorithm is to minimize the completion time of all jobs. This one-machine scheduling problem considers that, associated to each job j, there are the following values (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.

At each node of the branch-and-bound tree the upper bound is computed using the algorithm of Schrage [23]. This algorithm gives priority to higher values of the tails (q_j) when scheduling released jobs. We break ties preferring jobs with larger processing times.

The lower bound is computed as in [6]. The value of the solution where preemption is allowed, 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.

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 algorithm of Taillard [26].

4.2 Local Search

In order to build a simple local search algorithm we need to design a neighborhood structure, the way to inspect the neighborhood of a given solution, and a procedure to evaluate the quality of each solution.

We use a neighborhood structure very similar to the NB neighborhood of Dell'Amico and Trubian [10] and the one of Balas and Vazacopoulos [4]. 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, sharing the same machine. Borrowing the nomination of Balas and Vazacopoulos [4] we speak of forward and backward moves over forward and backward critical pairs of operations. Let $L(i,j)$ denote the length of the critical path from node i to node j.

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) guarantee that all moves lead to feasible solutions [4].

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 than 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 described in [4].

4.3 GRASP_B&B

Let *runs* be the total number of runs, M the set of machines and $f(x)$ the makespan of a solution x. The procedure GRASP_B&B can be generally described by the pseudo-code in Fig. 2:

```
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 initialized or $f(x) < f^*$
(15)        $x^* := x$
(16)        $f^* := f(x)$
(17)  return $x^*$
```

Fig. 2. Pseudo-code of algorithm GRASP_B&B

5 Computational Results

We have tested the algorithm GRASP_B&B (coded in C) on a Pentium 4 CPU 2.80 GHz, on the benchmark instances abz5-9 [1], ft6, ft10, ft20 [14], la01-40 [17],

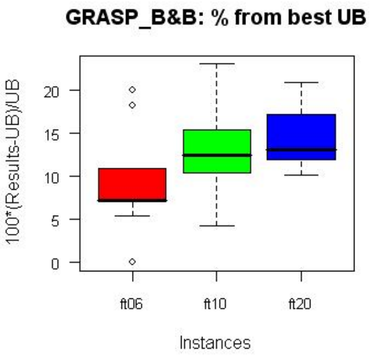

Fig. 3. Boxplots of RE_{UB} achieved with GRASP_B&B for the **ft** instances. ft06: (6*6); ft10: (10*10); ft20: (20*5).

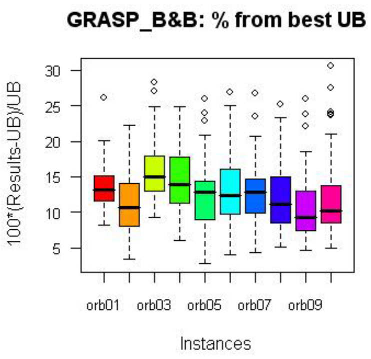

Fig. 4. Boxplots of RE_{UB} achieved with GRASP_B&B for the **orb** instances. orb01-10: (10*10).

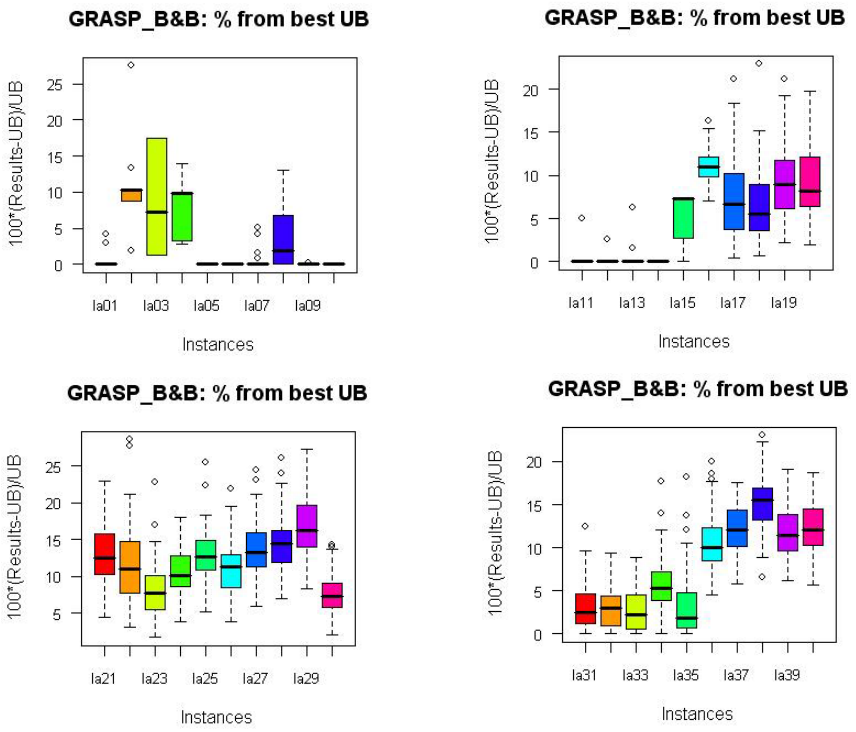

Fig. 5. Boxplots of RE_{UB} achieved with GRASP_B&B for the **la** instances. la01-05: (10*5); la06-10: (15*5); la11-15: (20*5); la16-20: (10*10); la21-25: (15*10); la26-30: (20*10); la31-35: (30*10); la36-40: (15*15).

orb01-10 [3], swv01-20 [24], ta01-70 [25] and yn1-4 [28]. The dimension of each instance is defined as the number of jobs times the number of machines ($n*m$).

Because of space limitations, in this work we will only present the results for instances ft6, ft10, ft20, la01-40 and orb01-10.

We show the results of running the algorithm 100 times for each instance presenting boxplots (figures 3 – 5) of RE_{UB}, the percentage of relative error to the best known upper bound (UB), calculated as follows:

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

We gathered the values of the upper bounds from [16], [20] and [21].

The boxplots 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, except maybe for instance la3. 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, except for instances ft06 and la38. On the other end, the worse values occur very seldom and are outliers for the majority of the instances.

Although this is a very simple (and fast) algorithm, the best values are not worse than the best known upper bound for 22 of the 152 instances used in this study.

5.1 Comparison to Other Algorithms

GRASP_B&B is a simple GRASP algorithm with a construction phase very similar to the one of the shifting bottleneck procedure. Therefore we show comparative results to two other procedures designed for the job shop problem; a simple GRASP procedure of Binato et al. [5] and the shifting bottleneck procedure of Adams et al. [1].

The building block of the construction phase of the GRASP in [5] is a single operation of a job. 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 time to the best solution found (btime) and the total time of all runs (ttime), in seconds. As the tables show, our algorithm is much faster. Whenever our GRASP_B&B achieves a solution not worse than theirs, we present the respective value in bold. This happens for 25 of the 53 instances whose results where compared.

Table 1. Comparing GRASP_B&B with (Binato et al 2001) and (Adams et al. 1988) - **ft** instances

name	GRASP_B&B	btime(s)	ttime (s)	GRASP	time (s)	Shifting Bottleneck	time (s)
ft06	**55**	0.1274	0.1400	55	70	55	1.5
ft10	<u>970</u>	0.5800	1.0000	938	261290	1015	10.1
ft20	<u>1283</u>	0.0094	0.4690	1169	387430	1290	3.5

Table 2. Comparing GRASP_B&B with (Binato et al 2001) and (Adams et al. 1988) - **la** instances

name	GRASP_B&B	btime (s)	ttime (s)	GRASP	time (s)	Shifting Bottleneck	time (s)
la01	**666**	0.0017	0.1720	666	140	666	1.26
la02	667	0.0437	0.1560	655	140	720	1.69
la03	605	0.0066	0.2190	604	65130	623	2.46
la04	607	0.0051	0.1710	590	130	597	2.79
la05	**593**	0.0011	0.1100	593	130	593	0.52
la06	**926**	0.0017	0.1710	926	240	926	1.28
la07	**890**	0.002	0.2030	890	250	890	1.51
la08	**863**	0.0149	0.2970	863	240	868	2.41
la09	**951**	0.0028	0.2810	951	290	951	0.85
la10	**958**	0.0014	0.1410	958	250	959	0.81
la11	**1222**	0.0027	0.2660	1222	410	1222	2.03
la12	**1039**	0.0027	0.2650	1039	390	1039	0.87
la13	**1150**	0.0038	0.3750	1150	430	1150	1.23
la14	**1292**	0.0022	0.2180	1292	390	1292	0.94
la15	**1207**	0.0453	0.9060	1207	410	1207	3.09
la16	1012	0.0221	0.7350	946	155310	1021	6.48
la17	787	0.0843	0.7660	784	60300	796	4.58
la18	854	0.3	0.7500	848	58290	891	10.2
la19	861	0.4554	0.9690	842	31310	875	7.4
la20	920	0.0813	0.8130	907	160320	924	10.2
la21	1092	0.1023	2.0460	1091	325650	1172	21.9
la22	**955**	0.9884	1.7970	960	315630	1040	19.2
la23	1049	1.7388	1.8900	1032	65650	1061	24.6
la24	**971**	0.627	1.8440	978	64640	1000	25.5
la25	**1027**	0.5388	1.7960	1028	64640	1048	27.9
la26	**1265**	3.0375	3.3750	1271	109080	1304	48.5
la27	**1308**	0.1781	3.5620	1320	110090	1325	45.5
la28	1301	0.15	3.0000	1293	110090	1256	28.5
la29	**1248**	0.857	3.2960	1293	112110	1294	48
la30	1382	0.8653	3.3280	1368	106050	1403	37.8
la31	**1784**	0.0702	7.0160	1784	231290	1784	38.3
la32	**1850**	0.5612	6.2350	1850	241390	1850	29.1
la33	**1719**	1.265	7.9060	1719	241390	1719	25.6
la34	**1721**	3.8093	8.2810	1753	240380	1721	27.6
la35	**1888**	0.2844	5.6880	1888	222200	1888	21.3
la36	**1325**	0.0853	4.2650	1334	115360	1351	46.9
la37	1479	4.0295	4.7970	1457	115360	1485	6104
la38	1274	0.7153	5.1090	1267	118720	1280	57.5
la39	1309	2.9835	4.4530	1290	115360	1321	71.8
la40	1291	3.5581	5.3910	1259	123200	1326	76.7

Table 3. Comparing GRASP_B&B with (Binato et al 2001) - **orb** instances

name	GRASP_B&B	btime (s)	ttime (s)	GRASP	time (s)
orb01	1145	0.0296	0.9850	1070	116290
orb02	918	0.0953	0.9530	889	152380
orb03	1098	0.335	1.0150	1021	124310
orb04	1066	0.8213	1.1250	1031	124310
orb05	911	0.105	0.8750	891	112280
orb06	1050	0.4812	1.0460	1013	124310
orb07	414	0.2764	1.0630	397	128320
orb08	945	0.3093	1.0310	909	124310
orb09	978	0.2809	0.9060	945	124310
orb10	991	0.2276	0.8430	953	116290

The comparison between the shifting bottleneck procedure [1] and the GRASP_B&B is presented in tables 1 and 2. Comparing the computation times of both procedures, 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 the respective value underlined. This happens in 25 of the 43 instances whose results where compared, and in 16 of the remaining 18 instances the best value found was the same.

6 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 [5], which is the base for a GRASP with path-relinking procedure [2], and the shifting bottleneck procedure, incorporated in the successful guided local search [4]. The comparison to the GRASP [5] shows that our procedure 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, ours is still faster, and it achieves better solutions, except for 2 of the comparable instances.

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. 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 (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.
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. and H.R. Lourenço (2006), "Optimized Search methods", Working paper, Universitat Pompeu Fabra, Barcelona, Spain.
14. 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.
15. Garey, M. R. and D. S. Johnson (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco, Freeman.
16. 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.
17. Lawrence, S. (1984), "Resource Constrained Project Scheduling: an Experimental Investigation of Heuristic Scheduling techniques", Graduate School of Industrial Administration, Carnegie-Mellon University.
18. 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.
19. 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.

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. Schrage, L. (1970). "Solving resource-constrained network problems by implicit enumeration: Non pre-emptive case." Operations Research, vol. 18: pp. 263-278.
24. 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.
25. Taillard, E. D. (1993). "Benchmarks for Basic Scheduling Problems." European Journal of Operational Research, vol. 64(2): pp. 278-285.
26. Taillard, É. D. (1994). "Parallel Taboo Search Techniques for the Job Shop Scheduling Problem." ORSA Journal on Computing, vol. 6(2): pp. 108-117.
27. 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.
28. 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.