COMBINING METAHEURISTICS FOR THE JOB SHOP SCHEDULING PROBLEM WITH SEQUENCE DEPENDENT SETUP TIMES

Miguel A. González, María R. Sierra, Camino R. Vela, Ramiro Varela, Jorge Puente
Artificial Intelligence Center, Dept. of Computing, University of Oviedo,
Campus de Viesques, 33271 Gijón, Spain

Keywords: Metaheuristics, Genetic Algorithms, Local Search, Job Shop Scheduling.

Abstract: The Job Shop Scheduling (JSS) is a hard problem that has interested to researchers in various fields such as Operations Research and Artificial Intelligence during the last decades. Due to its high complexity, only small instances can be solved by exact methods, while instances with a size of practical interest should be solved by means of approximate methods guided by heuristic knowledge. In this paper we confront the Job Shop Scheduling with Sequence Dependent Setup Times ($SDJSS$). The $SDJSS$ problem models many real situations better than the JSS. Our approach consists in extending a genetic algorithm and a local search method that demonstrated to be efficient in solving the JSS problem. We report results from an experimental study showing that the proposed approaches are more efficient than other genetic algorithm proposed in the literature, and that it is quite competitive with some of the state-of-the-art approaches.

1 INTRODUCTION

The Job Shop Scheduling Problem with Sequence Dependent Setup Times ($SDJSS$) is a variant of the classic Job Shop Scheduling Problem (JSS) in which a setup operation on a machine is required when the machine switches between two jobs. This way the $SDJSS$ models many real situations better than the JSS. The $SDJSS$ has interested to a number of researchers, so we can find a number of approaches in the literature, many of which try to extend solutions that were successful to the classic JSS problem. This is the case, for example, of the branch and bound algorithm proposed by Brucker and Thiele in (Brucker and Thiele, 1996), which is an extension of the well-known algorithms proposed in (Brucker et al., 1994), (Brucker, 2004) and (Carlier and Pinson, 1994), and the genetic algorithm proposed by Cheung and Zhou in (Cheung and Zhou, 2001), which is also an extension of a genetic algorithm for the JSS. Also, in (Zoghby et al., 2005) a neighborhood search with heuristic repairing is proposed that it is an extension of the local search methods for the JSS.

In this paper we apply a similar methodological approach and extend a genetic algorithm and a local search method that we have applied previously to the JSS problem. The genetic algorithm was designed by combining ideas taken from the literature such as for example the well-known $G\&T$ algorithm proposed by Giffler and Thomson in (Giffler and Thomson, 1960), the codification schema proposed by Bierwirth in (Bierwirth, 1995) and the local search methods developed by various researchers, for example Dell' Amico and Trubian in (Dell' Amico and Trubian, 1993), Nowicki and Smutnicki in (Nowicki and Smutnicki, 1996) or Mattfeld in (Mattfeld, 1995). In (González et al., 2006) we reported results from an experimental study over a set of selected problems showing that the genetic algorithm is quite competitive with the most efficient methods for the JSS problem.

In order to extend the algorithm to the $SDJSS$ problem, we have firstly extended the decoding algorithm, which is based on the $G\&T$ algorithm. Furthermore, in our local search method, we have adapted the neighborhood structure termed N_1 in the literature to obtain a neighborhood that we have termed N_1^S.

The experimental study was conducted over the set of 45 problem instances proposed by Cheung and Zhou in (Cheung and Zhou, 2001) and also over the set of 15 instances proposed by Brucker and Thiele in (Brucker and Thiele, 1996). We have evaluated the genetic algorithm alone and then in conjunction with

A. González M., R. Sierra M., R. Vela C., Varela R. and Puente J. (2006).
COMBINING METAHEURISTICS FOR THE JOB SHOP SCHEDULING PROBLEM WITH SEQUENCE DEPENDENT SETUP TIMES.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 211-218
DOI: 10.5220/0001312402110218

local search. The results show that the proposed genetic algorithm is more efficient than the genetic algorithm proposed in (Cheung and Zhou, 2001) and that the genetic algorithm combined with local search improves with respect to the raw genetic algorithm when both of them run during similar amount of time. Moreover, the efficiency of the genetic algorithm is at least comparable to the exact approaches proposed in (Brucker and Thiele, 1996) and (Artigues et al., 2004).

The rest of the paper is organized as it follows. In section 2 we formulate the $SDJSS$ problem. In section 3 we outline the genetic algorithm for the $SDJSS$. In section 4 we describe the extended local search method. Section 5 reports results from the experimental study. Finally, in section 6 we summarize the main conclusions.

2 PROBLEM FORMULATION

We start by defining the JSS problem. The classic JSS problem requires scheduling a set of N jobs $J_1, \ldots, J_N$ on a set of M physical resources or machines $R_1, \ldots, R_M$. Each job J_i consists of a set of tasks or operations $\{\theta_{i1}, \ldots, \theta_{iM}\}$ to be sequentially scheduled. Each task θ_{il} having a single resource requirement, a fixed duration $p\theta_{il}$ and a start time $st\theta_{il}$ whose value should be determined.

The JSS has two binary constraints: precedence constraints and capacity constraints. Precedence constraints, defined by the sequential routings of the tasks within a job, translate into linear inequalities of the type: $st\theta_{il} + p\theta_{il} \leq st\theta_{i(l+1)}$ (i.e. θ_{il} before $\theta_{i(l+1)}$). Capacity constraints that restrict the use of each resource to only one task at a time translate into disjunctive constraints of the form: $st\theta_{il}+p\theta_{il} \leq st\theta_{jk} \vee st\theta_{jk} + p\theta_{jk} \leq st\theta_{il}$. Where θ_{il} and θ_{jk} are operations requiring the same machine. The objective is to come up with a feasible schedule such that the completion time, i.e. the $makespan$, is minimized.

In the sequel a problem instance will be represented by a directed graph $G = (V, A \cup E)$. Each node in the set V represents a operation of the problem, with the exception of the dummy nodes $start$ and end, which represent operations with processing time 0. The arcs of the set A are called *conjunctive arcs* and represent precedence constraints and the arcs of set E are called *disjunctive arcs* and represent capacity constraints. Set E is partitioned into subsets E_i with $E = \cup_{i=1,\ldots,M} E_i$. Subset E_i corresponds to resource R_i and includes an arc (v, w) for each pair of operations requiring that resource. The arcs are weighed with the processing time of the operation at the source node. The dummy operation $start$ is connected to the first operation of each job; and the last operation of each job is connected to the node end.

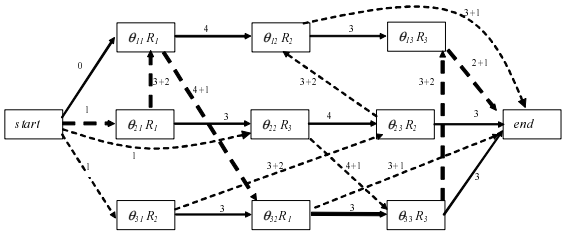

Figure 1: A feasible schedule to a problem with 3 jobs and 3 machines. Bold face arcs show a critical path whose length, i.e. the $makespan$, is 22.

A feasible schedule is represented by an acyclic subgraph G_s of G, $G_s = (V, A \cup H)$, where $H = \cup_{i=1..M} H_i$, H_i being a hamiltonian selection of E_i. Therefore, finding out a solution can be reduced to discovering compatible hamiltonian selections, i.e. orderings for the operations requiring the same resource or partial schedules, that translate into a solution graph G_s without cycles. The $makespan$ of the schedule is the cost of a *critical path*. A *critical path* is a longest path from node start to node end. The critical path is naturally decomposed into subsequences $B_1, \ldots, B_r$ called $critical\ blocks$. A critical block is a maximal subsequence of operations of a critical path requiring the same machine.

In the $SDJSS$, after an operation v of a job leaves machine m and before entering an operation w of another job on the same machine, a setup operation is required with duration S^m_{vw}. The setup operation can be started as soon as operation v leaves the machine m, hence possibly in parallel with the operation preceding w in its job sequence. The setup time S^m_{vw} is added to the processing time of operation v to obtain the cost of each disjunctive arc (v, w). S^m_{0v} is the setup time of machine m if v is the first operation scheduled on m and S^m_{v0} is the cleaning time of machine m if v is the last operation scheduled on m.

Figure 1 shows a feasible solution to a problem with 3 jobs and 3 machines. Dotted arcs represent the elements of set E included in the solution, while conjunctive arcs are represented by continuous arrows.

3 GENETIC ALGORITHM FOR THE SDJSS PROBLEM

The JSS is a paradigm of constraint satisfaction problems and was confronted by many heuristic techniques. In particular genetic algorithms (Bierwirth, 1995),(Mattfeld, 1995),(Varela et al., 2003),(González et al., 2006) are a promising approach due to their ability to be combined with other

Algorithm 1 Conventional Genetic Algorithm.

input: a JSS problem P
output: a schedule H for problem P
1. Generate the Initial Population;
2. Evaluate the Population;
while No termination criterion is satisfied **do**
3. Select chromosomes from the current population;
4. Apply the Crossover and Mutation operators to the chromosomes selected at step 3. to generate new ones;
5. Evaluate the chromosomes generated at step 4;
6. Apply the Acceptation criterion to the set of chromosomes selected at step 3. together with the chromosomes generated at step 4.;
end while
7. Return the schedule from the best chromosome evaluated so far;

techniques such as tabu search and simulated annealing. Moreover genetic algorithms allow for exploiting any kind of heuristic knowledge from the problem domain. In doing so, genetic algorithms are actually competitive with the most efficient methods for JSS.

As mentioned above, in this paper we consider a conventional genetic algorithm for tackling the JSS and extend it to the $SDJSS$. This requires mainly the adaptation of the decoding algorithm. Additionally we consider a local search method for the JSS and adapt it to the $SDJSS$.

Algorithm 1 shows the structure of the genetic algorithm we have considered. In the first step the initial population is generated and evaluated. Then the genetic algorithm iterates over a number of steps or generations. In each iteration a new generation is built from the previous one by applying the genetic operators of selection, crossover, mutation and acceptation. In principle, these four operators can be implemented in a variety of ways and are independent each one to the others. However in practice all of them should be chosen considering their effect on the remaining ones in order to get a successful convergence. The approach taken in this work is the following. In the selection phase all chromosomes are grouped into pairs, and then each one of these pairs is mated and mutated accordingly to the corresponding probabilities to obtain two offsprings. Finally a tournament selection is done among each pair of parents and their offsprings.

To codify chromosomes we have chosen permutations with repetition proposed by C. Bierwirth in (Bierwirth, 1995). In this scheme a chromosome is a permutation of the set of operations, each one being represented by its job number. This way a job number appears within a chromosome as many times as the number of operations of its job. For example, the chromosome (2 1 1 3 2 3 1 2 3) actually represents the permutation of operations $(\theta_{21}\,\theta_{11}\,\theta_{12}\,\theta_{31}\,\theta_{22}\,\theta_{32}\,\theta_{13}\,\theta_{23}\,\theta_{33})$. This permutation should be understood as expressing partial schedules for every set of operations requiring the same machine. This codification presents a number of interesting characteristics; for example, it is easy to evaluate with different algorithms and allows for efficient genetic operators. In (Varela et al., 2005) this codification is compared with other permutation based codifications and demonstrated to be the best one for the JSS problem over a set of 12 selected problem instances of common use. For chromosome mating we have considered the *Generalized Order Crossover* (GOX) that works as it is shown in the following example. Let us consider that the two following chromosomes are selected as parents for crossover

Parent1 (1 2 3 3 2 1 1 3 2) Parent2 (3 3 2 3 1 1 2 2 1)

Firstly, a substring is selected from Parent1 and inserted in the Offspring at the same position as in this parent. Then the remaining positions of the Offspring are completed with genes from Parent2 after having removed the genes selected from Parent1. If the selected substring from Parent1 is the one marked with underlined characters, the resulting Offspring is

Offspring (3 2 3 3 2 1 1 1 2).

By doing so, GOX preserves the order and position of the selected substring from Parent1 and the relative order of the remaining genes from Parent2. The mutation operator simply selects and swaps two genes at random. In practice the mutation would not actually be necessary due to the GOX operator has an implicit mutation effect. For example the second 3 from Parent1 is now the third one in the Offspring.

3.1 Decoding Algorithm

As decoding algorithm we have chosen the well-known $G\&T$ algorithm proposed by Giffler and Thomson in (Giffler and Thomson, 1960) for the JSS and then we have made a natural extension for the $SDJSS$. The $G\&T$ algorithm is an active schedule builder. A schedule is active if one operation must be delayed when you want another one to start earlier. Active schedules are good in average and, what is most important, it can be proved that the space of active schedules contains at least an optimal one, that is, the set of active schedules is *dominant*. For these reasons it is worth to restrict the search to this space. Moreover, the $G\&T$ algorithm is complete for the JSS problem. Algorithm 2 shows the $G\&T$ algorithm for the JSS.

In order to adapt the $G\&T$ algorithm for the $SDJSS$ we consider an extension termed $EG\&T$.

Algorithm 2 The decoding Giffler and Thomson algorithm for the JSS problem .

input: a chromosome C and a problem P
output: the schedule H represented by chromosome C for problem P
1. A = set containing the first operation of each job;
while $A \neq \emptyset$ **do**
2. Determine the operation $\theta' \in A$ with the earliest completion time if scheduled in the current state, that is $st\theta' + p\theta' \leq st\theta + p\theta, \forall\theta \in A$;
3. Let R be the machine required by θ', and B the subset of A whose operations require R;
4. Remove from B every operation that cannot start at a time earlier than $st\theta' + p\theta'$;
5. Select $\theta^* \in B$ so that it is the leftmost operation of B in the chromosome sequence;
6. Schedule θ^* as early as possible to build the partial schedule corresponding to the next state;
7. Remove θ^* from A and insert the succeeding operation of θ^* in set A if θ^* is not the last operation of its job;
end while
8. return the built schedule;

$EG\&T$ can be derived from the algorithm $EGTA1$ developed by Ovacik and Uzsoy in (Ovacik and Uzsoy, 1993), by simply taking into account the setup times in Algorithm 2. So, the step 4 of Algorithm 2 is exchanged by

4. Remove from B every operation θ that $st\theta \geq st\theta' + p\theta' + S^R_{\theta'\theta}$ for any $\theta' \in B$;

In Algorithm 2, $st\theta$ refers to the maximum completion time of the last scheduled operation on the machine required by operation θ and the preceding operation to θ in its job. Hence the algorithm can be adapted to the $SDJSS$ problem by considering $st\theta$ as the maximum completion time of the preceding operation in the job and the completion time of the last scheduled operation in the machine plus the corresponding setup time. It is easy to demonstrate that $EG\&T$ is not complete. In (Artigues et al., 2005) two more extensions of the $G\&T$ schedule generation scheme are proposed, one of them is not complete either, and the other is complete but it is very time consuming due to it needs to do backtracking. In any case, the lack of completeness of a decoding algorithm is not a serious problem in the framework of GAs due to a GA itself is not complete. Moreover, the local search schema outlined in the next section gives to any chromosome the chance of being reached, so in any way the lack of completeness of the decoding algorithm is compensated.

Algorithm 3 The Local Search Algorithm.

input: a chromosome C and a JSS problem P
output: a (hopefully) improved chromosome
1. Evaluate chromosome C (Algorithm 2) to obtain schedule H;
while No termination criterion is satisfied **do**
2. Generate the neighborhood of H with some method N, $N(H)$;
3. Select $H' \in N(H)$ with the selection criterion;
4. Replace H by H' if the acceptation criterion holds;
end while
5. Rebuild chromosome C from schedule H;
6. return chromosome C;

4 LOCAL SEARCH

Conventional genetic algorithms, like the one described in the previous section, often produce moderate results. However meaningful improvements can be obtained by means of hybridization with other methods. One of such techniques is local search, in this case the genetic algorithm is called a memetic algorithm. Hybridization of a genetic algorithm with local search is carried out by applying the local search algorithm to every chromosome just after this chromosome is generated, instead of simply applying the Algorithm 2 as it is done in the simple genetic algorithm. Algorithm 3 shows the typical strategy of a local search.

Roughly speaking local search is implemented by defining a neighborhood of each point in the search space as the set of chromosomes reachable by a given transformation rule. Then a chromosome is replaced in the population by one of its neighbors, if any of them satisfies the acceptation criterion. The local search from a given point completes either after a number of iterations or when no neighbor satisfies the acceptation criterion.

In this paper we consider the neighborhood structure proposed by Nowicki and Smutnicki in (Nowicki and Smutnicki, 1996), which is termed N_1 by D. Mattfeld in (Mattfeld, 1995), for the JSS. As other strategies, N_1 relies on the concepts of critical path and critical block. It considers every critical block of a critical path and made a number of moves on the operations of each block. After a move inside a block, the feasibility must be tested. Since an exact procedure is computationally prohibitive, the feasibility is estimated by an approximate algorithm proposed by Dell' Amico and Trubian in (Dell' Amico and Trubian, 1993). This estimation ensures feasibility at the expense of omitting a few feasible solutions. In (Mattfeld, 1995) the transformation rules of N_1 are defined

as follows.

Definition 1 (N_1) *Given a schedule H with partial schedules H_i for each machine R_i, $1 \leq i \leq M$, the neighborhood $N_1(H)$ consist of all schedules derived from H by reversing one arc (v, w) of the critical path with $(v, w) \in H_i$. At least one of v and w is either the first or the last member of a block. For the first block only v and w at the end of the block are considered whereas for the last block only v and w at the beginning of the block must be checked.*

The selection strategy of a neighbor and the acceptation criterion are based on a $makespan$ estimation, which is done in constant time as it is also described in (Dell' Amico and Trubian, 1993), instead of calculating the exact $makespan$ of each neighbor. The estimation provides a lower bound of the $makespan$. The selected neighbor is the one with the lowest $makespan$ estimation whenever this value is lower than the $makespan$ of the current chromosome. Notice that this strategy is not steepest descendent because the exact $makespan$ of selected neighbor is not always better than the $makespan$ of the current solution. We have done this choice in the classic JSS problem due to it produces better results than a strict steepest descent gradient method. (González et al., 2006).

The Algorithm stops either after a number of iterations or when the estimated $makespan$ of selected neighbor is larger than the $makespan$ of the current chromosome.

This neighborhood relies on the fact that, for the JSS problem, reversing an arc of the critical path always maintains feasibility. Moreover, the only possibility to obtain some improvement by reversing an arc is that the reversed arc is either the first or the last of a critical block.

However, things are not the same for $SDJSS$ problem due to the differences in the setup times. As can we see in (Zoghby et al., 2005), feasibility is not guaranteed when reversing an arc of the critical path, and reversing an arc inside a block could lead to an improving schedule. The following results give sufficient conditions of no-improving when an arc is reversed in a solution H of the $SDJSS$ problem. In the setup times the machine is omitted for simplicity due to all of them refers to the same machine.

Theorem 1 *Let H be a schedule and (v, w) an arc that is not in a critical block. Then reversing the arc (v, w) does not produce any improvement even if the resulting schedule is feasible.*

Theorem 2 *Let H be a schedule and (v, w) an arc inside a critical block, that is there exist arcs (x, v) and (w, y) belonging to the same block. Even if the schedule H' obtained from H by reversing the arc (v, w) is feasible, H' is not better than H if the following condition holds*

$$S_{xw} + S_{wv} + S_{vy} \geq S_{xv} + S_{vw} + S_{wy} \quad (1)$$

Theorem 3 *Let H be a schedule and (v, w) an arc in a critical path so that v is the first operation of the first critical block and z is the successor of w in the critical path and $M_w = M_z$. Even if reversing the arc (v, w) leaves to a feasible schedule, there is no improvement if the following condition holds*

$$S_{0w} + S_{wv} + S_{vz} \geq S_{0v} + S_{vw} + S_{wz} \quad (2)$$

Analogous, we can formulate a similar result if w is the last operation of the last critical block.

Hence we can finally define the neighborhood strategy for the $SDJSS$ problem as it follows

Definition 2 *(N_1^S) Given a schedule H, the neighborhood $N_1^S(H)$ consist of all schedules derived from H by reversing one arc (v, w) of the critical path provided that none of the conditions given in previous theorems 1, 2 and 3 hold.*

4.1 Feasibility Checking

Regarding feasibility, for the $SDJSS$ it is always required to check it after reversing an arc. As usual, we assume that the triangular inequality holds, what is quite reasonable in actual production plans, that is for any operations u,v and w requiring the same machine

$$S_{uw} \leq S_{uv} + S_{vw} \quad (3)$$

Then the following is a necessary condition for no-feasibility after reversing the arc (v, w).

Theorem 4 *Let H be a schedule and (v, w) an arc in a critical path, PJ_w the operation preceding w in its job and SJ_v the successor of v in its job. Then if reversing the arc (v, w) produces a cycle in the solution graph, the following condition holds*

$$stPJ_w > stSJ_v + duSJ_v + S_{min} \quad (4)$$

where

$$S_{min} = min\{S_{kl}/(k, l) \in E, J_k = J_v\}$$

and J_k is the job of operation k.

Therefore the feasibility estimation is efficient at the cost of discarding some feasible neighbor.

4.2 Makespan Estimation

For $makespan$ estimation after reversing an arc, we have also extended the method proposed by Taillard in (Taillard, 1993) for the JSS. This method was used also by Dell'Amico and Trubian in (Dell' Amico and Trubian, 1993) and by Mattfeld in (Mattfeld, 1995).

This method requires calculating *heads* and *tails*. The head r_v of an operation v is the cost of the longest path from node *start* to node v in the solution graph, i.e. is the value of stv. The tail q_v is defined so as the value $q_v + p_v$ is the cost of the longest path from node v to node *end*.

For every node v, the value $r_v + p_v + q_v$ is the length of the longest path from node *start* to node *end* trough node v, and hence it is a lower bound of the $makespan$. Moreover, it is the $makespan$ if node v belongs to the critical path. So, we can get a lower bound of the new schedule by calculating $r_v + p_v + q_v$ after reversing (v, w).

Let us denote by PM_v and SM_v the predecessor and successor nodes of v respectively on the machine sequence in a schedule. Let nodes x and z be PM_v and SM_w respectively in schedule H. Let us note that in H' nodes x and z are PM_w and SM_v respectively. Then the new heads and tails of operations v and w after reversing the arc (v, w) can be calculated as the following

$$r'_w = max(r_x + px + S_{xw}, r_{PJ_w} + pPJ_w)$$
$$r'_v = max(r'_w + pw + S_{wv}, r_{PJ_v} + pPJ_v)$$
$$q'_v = max(q_z + pz + S_{vz}, q_{SJ_v} + pSJ_v)$$
$$q'_w = max(q'_v + pv + S_{vw}, q_{SJ_w} + pSJ_w)$$

From these new values of heads and tails the $makespan$ of H' can be estimated by

$$C'_{max} = max(r'_v + pv + q'_v, r'_w + pw + q'_w)$$

which is actually a lower bound of the new $makespan$. This way, we can get an efficient $makespan$ estimation of schedule H' at the risk of discarding some improving schedule.

5 EXPERIMENTAL STUDY

For experimental study we have used the set of problems proposed by Cheung and Zhou in (Cheung and Zhou, 2001) and also the benchmark instances taken from Brucker and Thiele (Brucker and Thiele, 1996). The first one is a set of 45 instances with sizes (given by the number of jobs and number of machines $N \times M$) 10×10, 10×20 and 20×20, which is organized into 3 types. Instances of type 1 have processing times and setup times uniformly distributed in (10,50); instances of type 2 have processing times in (10,50) and setup times in (50,99); and instances of type 3 have processing times in (50,99) and setup times in (10,50). Table 1 shows the results from the genetic algorithm termed GA_SPTS reported in (Cheung and Zhou, 2001). The data are grouped for sizes and types and values reported are averaged for each group. This algorithm was coded in FORTRAN and run on PC 486/66. The computation time with problem sizes 10×10, 10×20 and 20×20 are about 16, 30 and 70 minutes respectively. Each algorithm run was stopped at the end of the 2000th generation and tried 10 times for each instance.

Table 1: Results from the GA_SPTS.

ZRD Instance	Size $N \times M$	Type	Best	Avg	StDev
1-5	10×10	1	835,4	864,2	21,46
6-10	10×10	2	1323,0	1349,6	21,00
11-15	10×10	3	1524,6	1556,0	35,44
16-20	20×10	1	1339,4	1377,0	25,32
21-25	20×10	2	2327,2	2375,8	46,26
26-30	20×10	3	2426,6	2526,2	75,90
31-35	20×20	1	1787,4	1849,4	57,78
36-40	20×20	2	2859,4	2982,0	93,92
41-45	20×20	3	3197,8	3309,6	121,52

Table 2: Results from the $GA\_EG\&T$.

ZRD Instances	Size $N \times M$	Type	Best	Avg	StDev
1-5	10×10	1	785,0	803,0	8,76
6-10	10×10	2	1282,0	1300,2	9,82
11-15	10×10	3	1434,6	1455,4	12,87
16-20	20×10	1	1285,8	1323,0	15,38
21-25	20×10	2	2229,6	2278,2	22,24
26-30	20×10	3	2330,4	2385,8	23,91
31-35	20×20	1	1631,6	1680,4	17,99
36-40	20×20	2	2678,0	2727,8	23,60
41-45	20×20	3	3052,0	3119,6	29,33

Table 3: Results from the $GA\_EG\&T\_LS$.

ZRD Instances	Size $N \times M$	Type	Best	Avg	StDev
1-5	10×10	1	778,6	788,5	6,70
6-10	10×10	2	1270,0	1290,4	9,16
11-15	10×10	3	1433,8	1439,8	6,71
16-20	20×10	1	1230,2	1255,5	12,74
21-25	20×10	2	2178,4	2216,8	18,61
26-30	20×10	3	2235,2	2274,0	19,32
31-35	20×20	1	1590,0	1619,8	15,90
36-40	20×20	2	2610,2	2668,0	27,48
41-45	20×20	3	2926,0	2982,2	26,32

Tables 2 and 3 reports the results reached by the genetic algorithm alone and the genetic algorithm with local search, termed $GA\_EG\&T$ and $GA\_EG\&T\_LS$ respectively, proposed in this work. In the first case the genetic algorithm was parameterized with a population of 100 chromosomes, a number of 140 generations, crossover probability of 0.7,

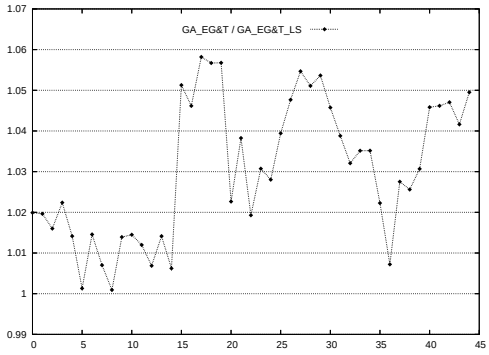

Figure 2: Comparison of the raw genetic algorithm with the memetic algorithm. The graphic shows for each problem the quotient of the mean makespan of the best solutions reached in all 30 trials by the raw GA and the GA with local search.

and mutation probability of 0.2. For the experiments combining the genetic algorithm with local search, we have parameterized the genetic algorithms with 50 chromosomes in the population and 50 generations in order to have similar running times.

The rest of the parameters remain as in previous experiments. The genetic algorithm was run 30 times and reported the values of the best solution reached, the average of the best solutions of the 30 runs and the standard deviation. The machine was a Pentium IV at 1.7 Ghz. and the computation time varied from about 1 sec. for the smaller instances to about 10 sec. for the larger ones. As we can observe both algorithms improved the results obtained by the GA_SPTS. Moreover algorithm $GA\_EG\&T\_LS$ has outperformed $GA\_EG\&T$. Figure 2 shows the relative improvement of $GA\_EG\&T\_LS$ over $GA\_EG\&T$ in all problems. The improvement is clear in almost all cases.

Regarding the benchmark from Brucker and Thiele (Brucker and Thiele, 1996), these instances are defined from the classical JSS instances, proposed by Lawrence (Lawrence, 1984), by introducing setup times. There are 15 instances named $t2\_ps01$ to $t2\_ps15$. Instances $t2\_ps01$ to $t2\_ps05$ are of type 10×5 (small instances). Instances $t2\_ps06$ to $t2\_ps10$ are of type 15×5 (medium instances). Instances $t2\_ps11$ to $t2\_ps15$ are of type 20×5 (large instances). Table 4 shows results from two state-of-the-art methods: the branch and bound algorithms proposed by Brucker and Thiele (Brucker and Thiele, 1996) (denoted as $BT96$) and Artigues et al. in (Artigues et al., 2004) (denoted as $ABF04$). In the results reported in (Brucker and Thiele, 1996) and (Artigues et al., 2004) the target machine was Sun 4/20 station and Pentium IV at 2.0 GHz. in both cases the time limit for the experiments was 7200 sec. In this case, our

Table 4: Comparison between $BT96$, $ABF04$ and $GA\_EG\&T\_LS$.

Problem Instance	Size $N \times M$	$BT96$	$ABF04$	$GA\_EG\&T\_LS$
$t2\_ps01$	10×5	**798**	**798**	**798**
$t2\_ps02$	10×5	**784**	**784**	**784**
$t2\_ps03$	10×5	**749**	**749**	**749**
$t2\_ps04$	10×5	**730**	**730**	**730**
$t2\_ps05$	10×5	**691**	**691**	693
$t2\_ps06$	15×5	1056	1026	1026
$t2\_ps07$	15×5	1087	**970**	**970**
$t2\_ps08$	15×5	1096	1002	975
$t2\_ps09$	15×5	1119	1060	1060
$t2\_ps05$	15×5	1058	**1018**	**1018**
$t2\_ps06$	20×5	1658	-	1450
$t2\_ps07$	20×5	1528	1319	1347
$t2\_ps08$	20×5	1549	1439	1431
$t2\_ps09$	20×5	1592	-	1532
$t2\_ps05$	20×5	1744	-	1523

values in bold are optimal

memetic algorithm was parameterized as the following: population size = 100 for small and medium instances and 200 for larger instances, and the number of generations has been 100 for small instances, 200 for medium instances, and 400 for larger instances. The rest of the parameters remain as in previous experiments. We run the algorithm 30 times for each instance, and the computation time for the larger instances was 30 sec. for each run, i.e. 900 sec. of running time for each instance.

As we can observe, $GA\_EG\&T\_LS$ is able to reach optimal solutions for the smaller instances, as $BT96$ and $ABF04$, with only one exception. For the medium and large instances reaches solutions that are better or equal than $ABF04$ and much better that $BT06$. Unfortunately, for the larger instances, results from only two instances are reported in (Artigues et al., 2004).

6 CONCLUSION

In this work we have confronted the Job Shop Scheduling Problem with Sequence Dependent Setup Times by means of a genetic algorithm hybridized with local search. As other approaches reported in the literature, we have extended a solution developed for the classic JSS problem. We have reported results from an experimental study on the benchmark proposed in (Cheung and Zhou, 2001) showing that the proposed genetic algorithms produce better results than the genetic algorithm proposed in (Cheung and Zhou, 2001), mainly when these algorithms are hy-

bridized with local search. Here it is important to remark that the running conditions of both genetic algorithms are not strictly comparable. Also we have experimented with the benchmark proposed by Brucker and Thiele in (Brucker and Thiele, 1996), and compare our memetic algorithm with two state-of-the-art exact branch and bound approaches due to Brucker and Thiele (Brucker and Thiele, 1996) and Artigues et al. in (Artigues et al., 2004) respectively. In this case the results shown that our approach is quite competitive.

As future work we plan to look for new extensions of the $G\&T$ algorithm in order to obtain a complete decoding algorithm and more efficient operators. Also we will try to extend other local search algorithms and neighborhoods that have been proved to be very efficient for the JSS problem.

ACKNOWLEDGEMENTS

We would like to thank Waiman Cheung and Hong Zhou, and Christian Artigues for facilitating us the benchmarks used in the experimental study. This research has been supported by FEDER-MCYT under contract TIC2003-04153 and by FICYT under grant BP04-021.

REFERENCES

Artigues, C., Belmokhtar, S., and Feillet, D. (2004). *A New Exact Algorithm for the Job shop Problem with Sequence Dependent Setup Times*, pages 96–109. LNCS 3011. Springer-Verlag.

Artigues, C., Lopez, P., and P.D., A. (2005). Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis. *Annals of Operational Research*, 138:21–52.

Bierwirth, C. (1995). A generalized permutation approach to jobshop scheduling with genetic algorithms. *OR Spectrum*, 17:87–92.

Brucker, P. (2004). *Scheduling Algorithms*. Springer-Verlag, 4th edition.

Brucker, P., Jurisch, B., and Sievers, B. (1994). A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107–127.

Brucker, P. and Thiele, O. (1996). A branch and bound method for the general-job shop problem with sequence-dependent setup times. *Operations Research Spektrum*, 18:145–161.

Carlier, J. and Pinson, E. (1994). Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:146–161.

Cheung, W. and Zhou, H. (2001). Using genetic algorithms and heuristics for job shop scheduling with sequence-dependent setup times. *Annals of Operational Research*, 107:65–81.

Dell' Amico, M. and Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operational Research*, 41:231–252.

Giffler, B. and Thomson, G. (1960). Algorithms for solving production scheduling problems. *Operations Reseach*, 8:487–503.

González, M., Sierra, M., Vela, C., and Varela, R. (2006). *Genetic Algorithms Hybridized with Greedy Algorithms and Local Search over the Spaces of Active and Semi-active Schedules*. LNCS (to appear). Springer-Verlag.

Lawrence, S. (1984). Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). *Technical report, Graduate School of Industrial Administration, Carnegie Mellon University*.

Mattfeld, D. (1995). *Evolutionary Search and the Job Shop. Investigations on Genetic Algorithms for Production Scheduling*. Springer-Verlag.

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

Ovacik, I. and Uzsoy, R. (1993). Exploiting shop floors status information to schedule complex jobs. *Operations Research Letters*, 14:251–256.

Taillard, E. (1993). Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal of Computing*, 6:108–117.

Varela, R., Serrano, D., and M., S. (2005). *New Codification Schemas for Scheduling with Genetic Algorithms*, pages 11–20. LNCS 3562. Springer-Verlag.

Varela, R., Vela, C., Puente, J., and A., G. (2003). A knowledge-based evolutionary strategy for scheduling problems with bottlenecks. *European Journal of Operational Research*, 145:57–71.

Zoghby, J., Barnes, J., and J.J., H. (2005). Modeling the re-entrant job shop scheduling problem with setup for metaheuristic searches. *European Journal of Operational Research*, 167:336–348.