A new Hybrid Genetic Algorithm for the Job Shop Scheduling Problem with Setup Times

Abstract

In this paper we face the Job Shop Scheduling Problem with Sequence Dependent Setup Times by means of a genetic algorithm hybridized with local search. We have built on a previous work and propose a new neighborhood structure for this problem which is based on reversing operations on a critical path. We have conducted an experimental study across the conventional benchmarks and some new ones of larger size. The results of these experiments show clearly that our approach outperforms the current state-of-the-art methods.

Introduction

In the last decades, scheduling problems have been subject to intensive research due to their multiple applications in areas of industry, finance and science (Brucker & Knust 2006). In this paper, we consider the Job Shop Scheduling Problem with Sequence Dependent Setup Times (SDST-JSP). This is an extension of the classical Job Shop Scheduling Problem (JSP) in which a setup operation is required between any two consecutive operations on the same machine. In this way, the SDST-JSP models much more real situations than the JSP. At the same time, setup considerations makes the problem more complex so as formal properties and methods have to be reconsidered. The SDST-JSP has been confronted by a number of researchers. Consequently, a number of approaches can be found in the literature that, in general, try to extend solutions that were successful for the classical JSP. For example, the branch and bound algorithm proposed by (Brucker & Thiele 1996) is an extension of the well-known algorithms proposed in (Brucker, Jurisch, & Sievers 1994), (Brucker 2004) and (Carlier & Pinson 1994). The genetic algorithm proposed by (Cheung & Zhou 2001) is also an extension of a genetic algorithm for the JSP. Also, (Balas, Simonetti, & Vazacopoulos 2005) extend the shifting bottleneck heuristic proposed in (Adams, Balas, & Zawack 1988) for the JSP.

In this paper we follow a similar methodological approach and extend a genetic algorithm and a local search method that were successfully applied to the JSP in (— 2006b). This genetic algorithm was designed by combining ideas taken from the literature such as the $G\&T$ algorithm proposed in (Giffler & Thomson 1960) and the codification schema proposed by (Bierwirth 1995). The local search methods are inspired by those developed for the JSP by various researchers, for example (Dell' Amico & Trubian 1993), (Nowicki & Smutnicki 1996), (Mattfeld 1995) or (Jain, Rangaswarny, & Meeran 2000). In (— 2006b) we report results from an experimental study across a set of hard problems selected by (Applegate & Cook 1991) showing that the genetic algorithm hybridized with local search is competitive with the most efficient methods for the JSP. In order to extend this algorithm to the SDST-JSP, we have first substituted the decoding algorithm by the Serial Schedule Generation Schema proposed in (Artigues, Lopez, & Ayache 2005). Then, for local search we have envisaged a new neighborhood structure which is inspired by the branching schema of the $B\&B$ algorithm proposed by (Brucker, Jurisch, & Sievers 1994) for the JSP and also by some ideas taken from (Dell' Amico & Trubian 1993) and (Nowicki & Smutnicki 1996). The new hybrid genetic algorithm is an extension of the algorithm proposed in (— 2008).

For the purpose of comparison with other methods, the experimental study has been conducted over the set of 15 instances proposed by (Brucker & Thiele 1996) (called BT set in the literature). First, we have evaluated separately the plain genetic algorithm and the local search starting from random solutions. Then, we have combined both methods; our approach consists of applying the local search to every chromosome generated by the genetic algorithm. The reported results show that the genetic algorithm and the local search produce quite similar results and that the genetic algorithm combined with local search is much more efficient than either of them alone, when all of them are run for the same or similar amount of time. Moreover, the results show that solutions obtained by the hybrid algorithm are quite competitive with the results obtained by the best current approaches, such as the branch and bound method proposed by (Artigues & Feillet 2008), the heuristic method proposed by (Balas, Simonetti, & Vazacopoulos 2005) or the hybrid genetic algorithm proposed in (— 2008). For the five larger instances we have even improved the best results reported so far. Additionally, we reported results of experiments conducted over another benchmark, proposed in (—

2008), with larger instances than that in the BT set. These instances were generated from the set of problems selected in (Applegate & Cook 1991) as hard to solve for the JSP.

The remaining of the paper is organized as follows. In the next section, we formulate the SDST-JSP and introduce the notation used across the paper. In the third section, we describe the genetic algorithm for the SDST-JSP, in particular we describe the local search algorithm and formalize the neighborhood structures for the SDST-JSP. The fourth section reports results obtained from the experimental study. Finally, in the last section we summarize the main conclusions.

The Job Shop Scheduling Problem with Sequence-Dependent Setup Times

The Job Shop Scheduling Problem with Sequence-Dependent Setup Times (SDST-JSP) 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_{ij} having a single resource requirement, a fixed duration $p_{\theta_{ij}}$ and a start time $st_{\theta_{ij}}$ whose value should be determined.

After an operation θ_{ij} leaves the machine and before an operation θ_{kl} enters the same machine, a setup operation is required with duration $S_{\theta_{ij}\theta_{kl}}$. $S_{0\theta_{ij}}$ is the setup time required before θ_{ij} if this operation is the first one scheduled on the machine, analogously $S_{\theta_{ij}0}$ is the cleaning time after operation θ_{ij} if this is the last operation scheduled on the machine.

The SDST-JSP 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: $st_{\theta_{ij}} + p_{\theta_{ij}} \leq st_{\theta_{i(j+1)}}$ (i.e. θ_{ij} before $\theta_{i(j+1)}$). Capacity constraints that restrict the use of each resource to only one task at a time translate into disjunctive constraints of the form: $st_{\theta_{ij}} + p_{\theta_{ij}} + S_{\theta_{ij}\theta_{kl}} \leq st_{\theta_{kl}} \vee st_{\theta_{kl}} + p_{\theta_{kl}} + S_{\theta_{kl}\theta_{ij}} \leq st_{\theta_{ij}}$, where θ_{ij} and θ_{kl} are operations requiring the same machine. The objective is to obtain a feasible schedule such that the completion time of all jobs, i.e. the *makespan*, denoted C_{max}, is minimized. This problem is denoted by $J|s_{ij}|C_{max}$ according to the $\alpha|\beta|\gamma$ notation used in the literature.

The Disjunctive Graph model Representation

The Disjunctive Graph is a common representation model for scheduling problems. The definition of such graph depends on the particular problem. For the SDST-JSP, Disjunctive Graphs can be defined as follows. A problem instance may be represented by a directed graph $G = (V, A \cup E \cup I)$. Each node in the set V represents a task of the problem, with the exception of the dummy nodes *start* or 0 and *end* or $nm + 1$ which represent tasks with null processing times. For a task θ_{ij}, the label of the corresponding node will be $k = m(i-1) + j$. The arcs of set A are called *conjunctive arcs* and represent precedence constraints and the arcs of set E are called *disjunctive arcs* and represent

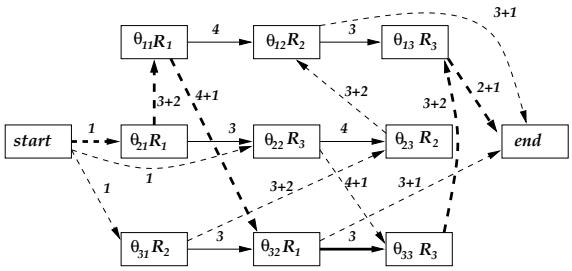

Figure 1: A feasible schedule to a problem with 3 jobs and 3 machines. Bold face arcs show a critical path whose length, i.e. the makespan, is 22.

capacity constraints. Set E is partitioned into subsets E_i, with $E = \cup_{i=1,\ldots,M} E_i$. E_i corresponds to resource R_i and includes an arc (v, w) for each pair of operations requiring that resource. Each arc (v, w) of A is weighted with the processing time of the operation at the source node, p_v, and each arc (v, w) of E is weighted with $p_v + S_{vw}$. The set I includes arcs of the form $(start, v)$ and (v, end) for each operation v of the problem. These arcs are weighted with S_{0v} and $p_v + S_{v0}$ respectively. In this paper, we consider that the triangular inequality holds for the setup times, i.e. $S_{uw} \leq S_{uv} + S_{vw}$ if u, v and w are operations requiring the same machine.

A feasible schedule is represented by an acyclic subgraph G_s of G, $G_s = (V, A \cup H \cup J)$, where $H = \cup_{i=1\ldots M} H_i$, H_i being a hamiltonian selection of E_i. J includes arcs $(start, v_i)$ and (w_i, end) for all $i = 1 \ldots M$, v_i and w_i being the first and last operations of H_i respectively.

Therefore, finding a solution can be reduced to discovering compatible hamiltonian selections, i.e. processing 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*. Nodes and arcs in a critical path are termed *critical*. 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.

Figure 1 shows a solution to a problem with 3 jobs and 3 machines. Dotted arcs represent the elements H and J, while arcs of A are represented by continuous arrows.

The concepts of critical path and critical block are of major importance for scheduling problems due to the fact that most of the formal properties and solution methods rely on them. For example, it is well-known that a schedule of a JSP instance is optimal if a critical path consists solely of operations of the same job or operations requiring the same machine. This is an interesting property, as it can be used as a termination criterion for a search algorithm. However, for the SDST-JSP only one of these properties holds, as it is proved in the following two theorems.

Theorem 1 *Let H be a schedule of a SDST-JSP instance. Even if H has critical path where all operations require the*

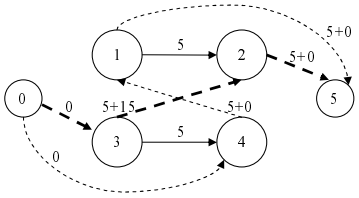

Figure 2: A feasible schedule to a problem with 2 jobs and 2 machines. Bold face arcs show a critical path whose length, i.e. the makespan, is 25.

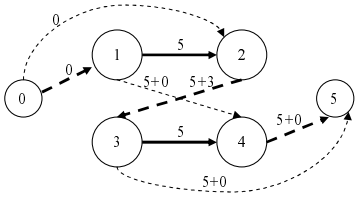

Figure 3: A feasible schedule to the same problem as 2. Bold face arcs show a critical path whose length, i.e. the makespan, is 23.

same machine, it might be non-optimal.

Proof 1 *Let us consider an instance with 2 jobs and 2 machines. θ_{11} and θ_{22} require R_1 and θ_{12} and θ_{21} require R_2. The processing time of the four operations is 5. All the setup times are zero, except for $S_{23} = 3$ and $S_{32} = 15$. It's clear that the triangular inequality holds for this instance. Figures 2 and 3 show a counterexample. Figure 2 shows a non-optimal schedule for this instance, with a critical path consisting of two operations requiring the same machine and Figure 3 shows the optimal schedule for this instance.*

Theorem 2 *Let H be a schedule of a SDST-JSP instance with a critical path where all operations belongs to the same job. If the triangular inequality holds for the setup times, then H is optimal.*

Proof 2 *Let i be the the job whose operations make up the critical path. Then the length of this path is $S_{0\theta_{i1}} + \sum_{j=1,\ldots,M} p_{\theta_{ij}} + S_{\theta_{iM}0}$. Due to the triangular inequality, θ_{i1} can not start at a time earlier than $S_{0\theta_{i1}}$ and the time after θ_{iM} can not be lower than $S_{\theta_{iM}0}$, so the length of that critical path is a lower bound of the optimal schedule and consequently H is optimal.*

The Genetic Algorithm

To solve the SDST-JSP we use here a variant of the hybrid genetic algorithm proposed in (— 2008). These algorithms differ mainly in the schedule builder schema used to evaluate chromosomes and also in the neighborhood structure used for local search.

To codify chromosomes, the schema based on permutations with repetition proposed in (Bierwirth 1995) is used. In this schema a chromosome is a permutation of the set of operations, each one being represented by its job number. In this way a job number appears within a chromosome as many times as the number of its operations. 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 each 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 efficient genetic operators. In (— 2005) this codification is compared with other permutation based codifications and demonstrated to be the best one for the JSP over a set of 12 selected problem instances of common use.

For chromosome mating, the genetic algorithm uses the *Job Order Crossover* (JOX) described in (Bierwirth 1995). Given two parents, JOX selects a random subset of jobs and copies their genes to the offspring in the same positions as they are in the first parent, then the remaining genes are taken from the second parent so as to maintain their relative ordering.

To build schedules, in (— 2008), we have used a decoding algorithm based on the algorithm $EG\&T$ proposed in (Artigues & Lopez 2000). This is an extension of the $G\&T$ algorithm for the JSP given in (Giffler & Thomson 1960) that considers setup times. Algorithm $EG\&T$ is simple and produces active schedules, but it is not dominant. In this paper, we consider a different schedule builder, the Serial Schedule Generation Scheme (SSGS) proposed by Artigues et al. in (Artigues, Lopez, & Ayache 2005). The schedule builder based on SSGS iterates over the operations in the order they appear in the chromosome sequence and for each one it selects the earliest starting time that satisfies all constraints with respect to the previous operations. Algorithm SSGS produces active schedules as well and it is dominant, provided that the triangular inequality holds. In some preliminary experiments, not reported here, we have compared SSGS with $EG\&T$ and SSGS have resulted more efficient overall.

Local Search

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 the selected neighbor, provided that it satisfies the acceptance criterion. In this paper, we use a simple hill-climbing based on makespan estimation, i.e. a chromosome is replaced by its first neighbor whose estimated makespan is better than the actual makespan of the chromosome. The local search from a given point finishes either after a number of iterations or when no neighbor satisfies the acceptance criterion. The local search algorithm is applied to every chromosome right after this chromosome has been generated.

In the neighborhood study carried out in this paper for the SDST-JSP, we have tried to extend the results and methods developed for the classical JSP. Some of the pioneer works for the JSP are described in (Matsuo, Suh, & Sullivan 1988)

and (Van Laarhoven, Aarts, & Lenstra 1992). In these papers, some interesting results are given such as that reversing a single critical arc always produce a feasible schedule and that reversing a single arc may produce an improvement only if the reversed arc is either at the beginning or at the end of a critical block. Also, for the JSP, the optimality of a schedule H can be established if the critical path contains only one critical block. From these results, a number of neighborhood structures have been proposed that gave rise to some of the most outstanding methods to solve the JSP. Among these we can cite (Dell' Amico & Trubian 1993), (Nowicki & Smutnicki 1996), (Nowicki & Smutnicki 2005), (Balas & Vazacopoulos 1998), (Zhang *et al.* 2008). Furthermore, a number of interesting properties have been studied and proved for some neighborhood structures. For example, some structures have the connectivity property; i.e. given any schedule H, an optimal one is always reachable from H by applying a finite sequence of transformations. This is the case of the structures proposed in (Van Laarhoven, Aarts, & Lenstra 1992) and (Dell' Amico & Trubian 1993). Should this property holds, the structure could also be exploited as a branching schema in an exact branch and bound algorithm. Also, for some structures, if the set of neighbors is empty for a schedule, this schedule is optimal. This is the case of the structure proposed in (Nowicki & Smutnicki 1996); this property allows to stop the search in about 20% of the instances solved in the experimental study.

For the SDST-JSP, things are not the same as for JSP problem. The problem structure changes significatively due to setup times, so as new approaches are required. Few results have been given with respect to neighborhood structures. For example, in (Zoghby, Barnes, & J.J. 2005), it is proved that feasibility is not guaranteed when reversing an arc of the critical path. Moreover, it easy to see that such a reversal might lead to improving schedules, even if this arc is not at the border of a critical block. In this paper, we establish two new results about connectivity and optimality for the SDST-JSP. This results are given in the following theorem.

Theorem 3 *Let N be a neighborhood structure such that $N(H)$ is made up of feasible schedules obtained from H by reversing processing orders of critical operations only. Then N does not fulfil the connectivity property. Moreover, H might not be globally optimal even if $N(H) = \emptyset$.*

Proof 3 *These results can be proved from the example presented in Figures 2 and 3. Let H be the schedule of figure 2. As we can observe, the only critical block of H is given by the arc $(3, 2)$, and reversing this arc gives rise to an unfeasible schedule, so $N(H) = \emptyset$. However, H is not optimal as it is worse than the schedule of figure 3 which is the only optimal solution for this problem. Hence, in order to reach the optimal solution from H, both a critical arc and a non-critical arc have to be simultaneously reversed.*

The connectivity is a convenient property for a neighborhood structure as, in principle, it reinforces the chance of success in finding and optimal schedule. However, it has not usually a great relevance in the context of a local search algorithm where the number of iterations is limited or the acceptation criterion is based on hill climbing.

In this paper we propose a new neighborhood structure for the SDST-JSP which is inspired by the branching schema used in the algorithm presented in (Brucker & Thiele 1996) for the SDST-JSP. This branching schema considers all critical blocks of a critical path, and for each block B, each operation is moved to any other position of B including before and after all the remaining operations of B. In order to establish an efficient neighborhood schema, we analyze all these moves and give sufficient conditions for non-improvement and feasibility. Also, we provide an algorithm for makespan estimation after a move.

Let us consider a critical block of the form $(b'\ v\ b\ w\ b'')$. In order to guarantee feasibility after moving the operation w before v, we have to proof that a cycle does not exist in the resulting solution graph. In other words, none of the alternative paths showed in figure 4 may exists. An exact determination of these paths not existing is very time consuming; for these reason it is usual to establish any sufficient condition, but not necessary, that can be efficiently evaluated. In this case we have the risk of discarding some feasible neighbors. In general, there is not only one sufficient condition so as we have to consider their cost and accuracy to chose the most appropriate one.

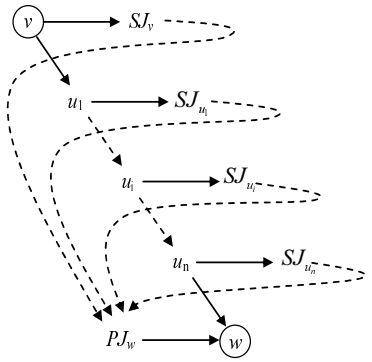

Figure 4: Potential alternative paths between two operations v and w in a critical block that could lead to a cycle after moving operation w before operation v.

The next result establishes a sufficient condition for feasibility after moving an operation w in a critical block towards the beginning of such block (just before v). This result is suitable for any structure involving reversing processing orders of tasks in a critical path, and have to be evaluated for each processing order reversed.

Theorem 4 *Given a critical block of the form $(b'\ v\ b\ w\ b'')$, where b, b' and b'' are sequences of operations, a sufficient condition for an alternative path from v to w not existing is that*

$$r_{PJ_w} < r_{SJ_u} + p_{SJ_u} + \min\{S_{kl}/(k,l) \in E, J_k = J_u\} \quad (1)$$
$$\forall u \in \{v\} \cup b$$

Proof 4 *Let us denote $b = (u_1 \dots u_i \dots u_n)$. The only alternative paths from v to w are those indicated in Figure 4. As each potential alternative path includes at least a setup time, none of these paths can exist if condition (1) holds.*

Here is important to remark that this condition can be evaluated in constant time for each pair of operations u and w and that all positions to which an operation w can be moved before its current position can be determined with only one iteration over operations preceding w in the critical block. An analogous reasoning can be done regarding moves toward the end of the critical block.

As we have commented above, after a neighbor is determined to be feasible, its makespan should be obtained in order to evaluate the acceptance condition of the local search. As obtaining the exact makespan for all neighbors is very time consuming, it is common to establish conditions for non-improvement that can be evaluated with a low cost. In this paper we give the following result that establishes a sufficient condition for non-improvement when an operation is moved inside a critical block.

Theorem 5 *Let H be a schedule and $(b'\ v\ b\ w\ b'')$ a critical block, where b, b' and b'' are sequences of operations of the form $b = (u_1 \dots u_n)$, $b' = (u'_1 \dots u'_{n'})$ and $b'' = (u''_1 \dots u''_{n''})$. Even if the schedule H' obtained from H by moving w just before v is feasible, H' does not improve H if the following condition holds*

$$S_{u_n v} + S_{u'_n w} + S_{w u''_1} \geq S_{u_n w} + S_{wv} + S_{u'_n u''_1}. \qquad (2)$$

In case of $n = 0$, u'_n should be substituted by v in (2).

Therefore, we can finally define the proposed neighborhood structure, termed N^S, as follows

Definition 1 **(N^S)** *Let operation v be a member of a critical block B. In a neighboring solution, v is moved to another position in B, provided that condition (2) is not fulfilled and that the sufficient condition of feasibility (1) is preserved.*

Makespan estimation For makespan estimation, we use the procedure *lpathS* given in (— 2008) for the SDST-JSP. This procedure takes as input a sequence of operations $(x\ Q_1 \dots Q_q\ y)$ requiring the same machine after a move, where $Q_1 \dots Q_q$ is a permutation of operations $O_1 \dots O_q$ appearing as $(x\ O_1 \dots O_q\ y)$ before the move. For each $i = 1 \dots q$, *lpathS* computes the cost of the shortest path from nodes $start$ to end through Q_i. The maximum of these values is taken as the makespan estimation for the neighboring schedule, which is clearly a lower bound.

This procedure is suitable for neighborhood structures proposed in (— 2008) and also for the structure proposed here by taking the appropriate input sequence $(Q_1 \dots Q_q)$ in each case. For N^S, if w is moved before v in a block of the form $(b'\ v\ b\ w\ b'')$, the input sequence is $(w\ v\ b)$.

Experimental Results

For experimental study we have used the set of problems proposed in (Brucker & Thiele 1996) (the BT set) and also some new benchmark instances proposed in (— 2008). The first one is a set of instances defined from the classical JSP instances proposed in (Lawrence 1984) by introducing setup times. Each instance is characterized by a number of machines, a number of jobs and a matrix of setup types of operations. So, each instance is identified by a triplet $(machines \times jobs \times setup\_types)$. There is a total of 15 instances named t2-ps01 to t2-ps15. Instances t2-ps01 to t2-ps05 are of type $5 \times 10 \times 5$ (small instances). Instances t2-ps06 to t2-ps10 are of type $5 \times 15 \times 5$ (medium instances). And instances t2-ps011 to t2-ps15 are of type $5 \times 20 \times 10$ (large instances). In (Brucker & Thiele 1996) and (Artigues & Feillet 2008) the authors remark that the corresponding Lawrence instances without setup times (LA01 to LA15) were easily solved by their branch and bound methods. We have also solved these instances to optimality with the genetic algorithm proposed in (— 2006a).

The paper by (Artigues & Feillet 2008) summarizes the best results obtained by the most efficient methods proposed so far. They report results from their branch and bound algorithm, as well as results from a number of previous approaches proposed by themselves and also from approaches proposed by other researches, such as the branch and bound method from (Brucker & Thiele 1996) and the method based on the shifting bottleneck heuristic from (Balas, Simonetti, & Vazacopoulos 2005).

Table 1 summarizes the best results reported in (Artigues & Feillet 2008) and also the results from (— 2008). It shows the best known lower bounds and the best solutions reached by three methods termed BSV05 (Balas, Simonetti, & Vazacopoulos 2005), AF08 (Artigues & Feillet 2008) and VVG08 (— 2008) respectively together with the times taken in any case. It is important to be aware of the differences in the target machines. The algorithm proposed in (Balas, Simonetti, & Vazacopoulos 2005) was implemented in C language and run on a Sun Ultra 60 with UltraSPARC-II processor at 360MHz. The hybrid GA from (— 2008) is coded in C++ and run in a Pentium IV (1.7GHz) machine. And the algorithm reported in (Artigues & Feillet 2008) is coded in C++ and run on a PC computer (no other details are given).

In the experimental study, we evaluated the local search and the genetic algorithm (GA) separately and both of them in combination. The results are summarized in Table 2. We run the GA 30 times for each problem instance and report the best solution reached in all 30 runs and the average of the best solutions from each run. The parameters are set to obtain similar run time in all cases. When the GA is combined with local search, it is parameterized as follows: for instances t2-ps01 to t2-ps05 the population size is 100 chromosomes and the number of generations is 100 (/100/100/), for instances of medium size t2-ps06 to t2-ps10 the parameters are /100/200/ and for the largest instances t2-ps10 to t2-ps15 the parameters are /200/400/. When local search is not used, the GA parameters are /190/190/, /270/400/ and /600/1000/ respectively. To study local search independently from GA, the local search algorithm is run 30 times for each instance starting from a set of 4000, 7000 and 25000 random solutions for small, medium and large instances respectively to obtain similar run times (about 0.78, 3.84 and 32.29 seconds per run for small, medium and large instances respectively). The algorithms are coded in C++ and the target ma-

Table 1: Summary of previous results

Instance	LB	BSV05		AF08		VVG08		
		Best	Time	(UB1)	Time	Best	Avg.	Time
t2-ps01	798	798*	358	798*	54	798*	799	2.7
t2-ps02	784	784*	550	784*	57	784*	784	2.6
t2-ps03	749	749*	834	749*	77	749*	749	2.3
t2-ps04	730	730*	515	730*	11	730*	732	2.1
t2-ps05	691	693	248	691*	14	691*	696	2.6
t2-ps06	1009	1018	1192	1009*	6151	1026	1026	13.3
t2-ps07	970	1003	1338	970*	10008	970*	970	10.8
t2-ps08	958	975	833	982	1471	963	974	12.4
t2-ps09	1051	1060	423	1061	595	1060	1061	10.2
t2-ps10	1018	1018*	557	1047	5692	1018*	1024	10.5
t2-ps11	1382	1470	3047	1494	8452	1441	1478	105.9
t2-ps12	1226	1305	2173	1381	1748	1293	1340	103.5
t2-ps13	1320	1439	2468	1457	12401	1415	1438	97.4
t2-ps14	1431	1485	2131	1483	3299	1466	1537	97.8
t2-ps15	1390	1527	3111	1661	25153	1492	1529	105.7

* tight bound
All algorithms are coded in C or C++ and target machines are a Sun Ultra 60 (360Mhz) in (BSV05); a PC computer in (AF08) and a Pentium IV (1.7GHz) in (VVG08) (in this case, times refers to average CPU time of an execution)

chine is Intel Core 2 Duo at 2,6 GHz.

As we can observe from the best and mean values in Table 2, the GA alone and the local search alone produce quite similar results, slightly better for the latter on the larger instances. Also, these methods are clearly worse than the GA combined with local search. The differences in efficiency can be more clearly appreciated from the mean relative errors also reported in Table 2, where the mean relative error is defined as $RE = 100 * (Average - BKS)/BKS$, being BKS the Best Known Solution. Also, the standard errors of the average values (calculated as $SE = \sigma/\sqrt{30}$, where σ is the standard deviation of the best solutions reached in the 30 runs) of local search algorithm are lower than those of the GA. These differences indicate that the average values from the local search algorithm are slightly more representative than the values from the GAs. However, if we consider the confidence interval with level 95% for the mean makespan from all three methods, we can see that the upper bound of the interval for the hybrid GA is much lower than the lower bounds for the local search and the GA. In summary, the efficiency of both methods is similar when they are exploited separately but synergies gained from the joint application largely improve the efficiency.

Regarding comparison with the branch and bound method described in (Artigues & Feillet 2008), the hybrid genetic algorithm achieves better mean values for seven instances, the same values for four instances, and worse values for the remaining four, as we can observe in Tables 1 and 2. It is worth noting that for four of the largest instances, the mean value of the hybrid genetic algorithm is better than the best value reached by the branch and bound algorithm. Moreover, in seven cases the mean value is also better than the best value reached by the shifting bottleneck heuristic described in (Balas, Simonetti, & Vazacopoulos 2005). Also, the new hybrid genetic algorithm outperforms the algorithm proposed in (— 2008), in particular regarding mean values. The standard error values confirm the reliability of these results, with a low sampling fluctuation on the average value obtained by the hybrid GA. Even for the large instance t2-ps12, with the highest standard error (3.81), the upper 95% confidence limit for the mean value is 1298.5 which is still better than the solutions given by (Artigues & Feillet 2008) and by (Balas, Simonetti, & Vazacopoulos 2005) (see Table 1).

Summarizing, the hybrid genetic algorithm reaches the optimal solution in all of the five small instances. For medium instances, it reaches the optimal solution in two cases and the best known solution in two cases. For instance t2-ps06 it is not able to reach the best known solution of 1009 which is optimal; however, in all 30 cases the solution reached was 1026 as it is in many of the methods reported in (Artigues & Feillet 2008). For the hardest instances, t2-ps11 to t2-ps15, the new hybrid genetic algorithm has improved the best known solutions in all cases.

Experiments with the new benchmark of selected instances

In order to experiment with problem instances larger than those in the BT set, we use the benchmark proposed in (— 2008). These instances are derived from the set of 10 selected problems identified in (Applegate & Cook 1991) as hard to solve for the classical JSP. The sizes of these problems are 15×10 for the smallest ones (La21 to La25), 20×10 for La27 and La29, 15×15 for La38 and La40, and 20×15 for the largest (ABZ instances). These instances are extended to the SDST-JSP by using the same criteria as in the BT instances, i.e. the setup times depend only on the jobs and are taken from one of two matrices. These matrices can be found, for example, in (Artigues, Lopez, & Ayache 2005)

Table 2: Summary of results from the comparison of the GA alone, Local Search alone and the GA hybridized with Local Search

	GA				LS				GA+LS				Time
Instance	Best	Avg.	RE	SE	Best	Avg.	RE	SE	Best	Avg.	RE	SE	sec.
t2-ps01	798*	806	0.96	1.60	798*	809	1.42	1.22	798*	798	0.00	0.00	0.70
t2-ps02	784*	786	0.19	0.43	784*	793	1.17	1.62	784*	784	0.00	0.00	0.73
t2-ps03	749*	762	1.67	1.35	749*	763	1.82	1.00	749*	749	0.03	0.33	0.81
t2-ps04	733	739	1.21	0.71	733	743	1.84	1.02	730*	730	0.00	0.00	0.70
t2-ps05	710	715	3.43	0.23	693	695	0.62	0.76	691*	692	0.13	0.57	0.96
t2-ps06	1026	1052	4.29	3.43	1026	1036	2.71	1.52	1026	1026	1.68	0.00	4.74
t2-ps07	988	1008	3.96	2.58	1020	1032	6.40	1.61	970*	971	0.08	0.00	3.34
t2-ps08	992	1023	6.18	2.30	988	1010	4.92	1.32	**963**	966	0.29	0.31	4.38
t2-ps09	1062	1075	1.41	1.59	1060	1069	0.89	1.35	**1060**	1060	0.00	0.00	3.54
t2-ps10	1018*	1052	3.31	4.28	1047	1062	4.29	1.12	1018*	1018	0.00	0.00	3.22
t2-ps11	1510	1564	8.75	7.04	1510	1526	6.10	1.45	**1438**[1]	1439[1]	0.05	0.39	31.62
t2-ps12	1357	1413	11.36	7.85	1350	1363	7.38	1.14	**1269**[1]	1291[1]	1.73	3.81	34.28
t2-ps13	1441	1510	7.42	6.44	1443	1472	4.71	2.59	**1406**[1]	1415	0.65	0.47	31.14
t2-ps14	1519	1581	8.86	6.92	1502	1539	6.00	1.51	**1452**[1]	1489	2.57	1.50	27.83
t2-ps15	1542	1602	7.87	4.91	1577	1612	8.58	2.90	**1485**[1]	1502	1.13	2.48	36.58

Values in **bold** are best known solutions, [1] improves previous best known solution, * tight bound.

Table 3: Summary of results across the benchmark issued from the 10 selected instances

	VVG08		GA+LS		
Instance	Best	Average	Best	Average	Time
La21_st	1351	1383	1351	1375	25.48
La24_st	1171	1177	1171	1177	22.48
La25_st	1221	1226	1213	1222	27.47
La27_st	1774	1802	1732	1770	80.24
La29_st	1740	1765	1729	1747	90.44
La38_st	1461	1473	1456	1469	42.59
La40_st	1470	1487	1476	1482	40.27
ABZ7_st	1273	1300	1242	1262	199.45
ABZ8_st	1294	1323	1255	1274	179.76
ABZ9_st	1264	1301	1235	1250	191.37

and identify the type of setup times 5 or 10. Instances with 15 jobs are of type 5 and instances with 20 jobs are type 10. Each instance is identified by the the name of the JSP instance followed by st. Table 3 summarizes the best and average values reached from both methods across these instances. We have set parameters for both hybrid GAs as /200/400/ for all instances. As before, the hybrid GA was run 30 times for each instance and the average and the best of the 30 solutions are reported. As parameters are different, results reported in Table3 for VVG08 method are, in general, better than those included in (— 2008). As we can see, the results reached with the new hybrid GA are also better than that obtained with the previous one.

Conclusions

We have considered the job shop problem with sequence dependent setup times, where the objective is to minimize the makespan. We have used a hybrid genetic algorithm, combining a genetic algorithm with a dominant active scheduler builder and a local search procedure. For local search, we have devised a new neighborhood structure which is based on reversing orders of operations on the critical path. We have studied the moves that might yield improving solutions in presence of setup times and we have established conditions that allow to discard unfeasible and some non-improving schedules. We have reported results from an experimental study across the benchmarks proposed in (Brucker & Thiele 1996) and (— 2008), and have compared our hybrid genetic algorithm with state-of-the-art methods. To our knowledge, the best of these methods are the branch and bound algorithm given by (Artigues & Feillet 2008) and the method based on the shifting bottleneck heuristic proposed in (Balas, Simonetti, & Vazacopoulos 2005). Also we have compared with another hybrid genetic algorithm from (— 2008) that differs in the schedule builder, the neighborhood structure and the acceptance criterion in the local search. The results of this study show that our approach outperforms all the current state-of-the-art methods.

As future work, we plan to study other moves in the critical path to establish new neighborhood structures and refine the conditions to check both feasibility and improvement. Also, we plan to experiment with other meta-heuristics such as tabu search or simulated annealing. These techniques have demonstrated to be very efficient for problems such as the classical JSP and we expect they will be good for the SDST-JSP as well.

Acknowledgments

References

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

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

Artigues, C., and Feillet, D. 2008. A branch and bound method for the job-shop problem with sequence-dependent setup times. *Annals of Operations Research* 159(1):135–159.

Artigues, C., and Lopez, P. 2000. Extending Giffler-Thompson algorithm to generate active schedules for job-shops with sequence-dependent setup times. *LIA report 129, University of Avignon*.

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

Balas, E., and Vazacopoulos, A. 1998. Guided local search with shifting bottleneck fo job shop scheduling. *Management Science* 44 (2):262–275.

Balas, E.; Simonetti, N.; and Vazacopoulos, A. 2005. Job shop scheduling with set-up times, deadlines and precedence constraints. In *Proceedings of MISTA'2005*.

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

Brucker, P., and Knust, S. 2006. *Complex Scheduling*. Springer.

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.

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. 2004. *Scheduling Algorithms*. Springer, 4th edition.

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 Operations 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. L. 1960. Algorithms for solving production scheduling problems. *Operations Research* 8:487–503.

— 2006a. Combining metaheuristics for the job shop scheduling problem with sequence dependent setup times. In *Proceedings of the First International Conference on Software and Data Techonoliges, ICSOFT'2006*, 211–220.

— 2006b. Genetic algorithms hybridized with greedy algorithms and local search over the spaces of active and semi-active schedules. *Lecture Notes in Computer Science* 4177:231–240.

Jain, A. S.; Rangaswarny, B.; and Meeran, S. 2000. New and "stronger" job-shop neighbourhoods: A focus on the method of nowicki and smutnicki (1996). *Journal of Heuristics* 6 (4):457–480.

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.

Matsuo, H.; Suh, C.; and Sullivan, R. 1988. A controlled search simulated annealing method for the general jobshop scheduling problem. *Working paper 03-44-88, Graduate School of Business, University of Texas*.

Mattfeld, D. C. 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 scheduling problem. *Management Science* 42:797–813.

Nowicki, E., and Smutnicki, C. 2005. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling* 8:145–159.

Van Laarhoven, P.; Aarts, E.; and Lenstra, K. 1992. Job shop scheduling by simulated annealing. *Operations Research* 40:113–125.

— 2005. New codification schemas for scheduling with genetic algorithms. *Lecture Notes in Computer Science* 3562:11–20.

— 2008. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics* Accepted with minor revisions.

Zhang, C. Y.; Li, P.; Rao, Y.; and Guan, Z. 2008. A very fast ts/sa algorithm for the job shop scheduling problem. *Computers and Operations Research* 35:282–294.

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.