An Efficient Memetic Algorithm for the Flexible Job Shop with Setup Times

Miguel A. González and **Camino R. Vela** and **Ramiro Varela**
Artificial Intelligence Center, University of Oviedo, Spain,
Campus of Viesques, 33271 Gijón
email {mig,crvela,ramiro}@uniovi.es

Abstract

This paper addresses the flexible job shop scheduling problem with sequence-dependent setup times (SDST-FJSP). This is an extension of the classical job shop scheduling problem with many applications in real production environments. We propose an effective neighborhood structure for the problem, including feasibility and non improving conditions, as well as procedures for fast neighbor estimation. This neighborhood is embedded into a genetic algorithm hybridized with tabu search. We conducted an experimental study to compare the proposed algorithm with the state-of-the-art in the SDST-FJSP and also in the standard FJSP. In this study, our algorithm has obtained better results than those from other methods. Moreover, it has established new upper bounds for a number of instances.

1 Introduction

The Job Shop Scheduling Problem (JSP) is a simple model of many real production processes. However, in many environments the production model has to consider additional characteristics or complex constraints. For example, in automobile, printing, semiconductor, chemical or pharmaceutical industries, setup operations such as cleaning up or changing tools are required between two consecutive jobs on the same machine. These setup operations depend on both the outgoing and incoming jobs, so they cannot be considered as being part of any of these jobs. Also, the possibility of selecting alternative routes among the machines is useful in production environments where multiple machines are able to perform the same operation (possibly with different processing times), as it allows the system to absorb changes in the demand of work or in the performance of the machines. When these two factors are considered simultaneously, the problem is known as the flexible job shop scheduling problem with sequence-dependent setup times (SDST-FJSP).

The classical JSP with makespan minimization has been intensely studied, and many results have been established that have given rise to very efficient algorithms, as for example the tabu search proposed in (Nowicki and Smutnicki 2005).

Incorporating sequence-dependent setup times or a flexible shop environment changes the nature of scheduling problems, so these well-known results and techniques for the JSP are not directly applicable. There are several papers dealing with each of the two factors (setups and flexibility). For example, the job shop with setups is studied in (Brucker and Thiele 1996), where the authors developed a branch and bound algorithm, and in (Vela, Varela, and González 2010) and (González, Vela, and Varela 2012) where the authors took some ideas proposed in (Van Laarhoven, Aarts, and Lenstra 1992) as a basis for new neighborhood structures.

The flexible job shop has been widely studied. Based on the observation that FJSP turns into the classical job shop scheduling problem when a machine assignment is chosen for all the tasks, early literature proposed hierarchical strategies for this complex scheduling problem, where machine assignment and sequencing are studied separately, see for example (Brandimarte 1993). However, more recent integrated approaches such as the tabu search algorithm proposed in (Mastrolilli and Gambardella 2000), the hybrid genetic algorithm proposed in (Gao, Sun, and Gen 2008) or the discrepancy search approach proposed in (Hmida et al. 2010), usually obtain better results.

However, very few papers have considered both flexibility and setup times at the same time. Among these, (Saidi-Mehrabad and Fattahi 2007), where the authors solve the problem with a tabu search algorithm or (Oddi et al. 2011), where the problem is solved by means of iterative flattening search, deserve special mention.

In this paper, we propose a new neighborhood structure for the SDST-FJSP. We also define feasibility and non improving conditions, as well as algorithms for fast estimation of neighbors' quality. This neighborhood is then exploited in a tabu search algorithm, which is embedded in a genetic algorithm. We conducted an experimental study in which we first analyzed the synergy between the two metaheuristics, and then we compared our algorithm with the state-of-the-art in both the SDST-FJSP and the standard FJSP.

The remainder of the paper is organized as follows. In Section 2 we formulate the problem and describe the solution graph model. In Section 3 we define the proposed neighborhood. Section 4 details the metaheuristics used. In Section 5 we report the results of the experimental study, and finally Section 6 summarizes the main conclusions of this paper and give some ideas for future work.

2 Problem formulation

In the job shop scheduling problem (JSP), we are given a set of n jobs, $J = \{J_1, \dots, J_n\}$, which have to be processed on a set of m machines or resources, $M = \{M_1, \dots, M_m\}$. Each job J_j consists of a sequence of n_j operations (θ_{j1}, θ_{j2}, …, θ_{jn_j}), where θ_{ij} must be processed without interruption on machine $m_{ij} \in M$ during $p_{ij} \in \mathbb{N}$ time units. The operations of a job must be processed one after another in the given order (the job sequence) and each machine can process at most one operation at a time.

We also consider the addition of sequence-dependent setup times. Therefore, if we have two operations u and v (we use this notation to simplify expressions), a setup time s_{uv} is needed to adjust the machine when v is processed right after u. These setup times depend on both the outgoing and incoming operations. We also define an initial setup time of the machine required by v, denoted s_{0v}, required when this is the first operation on that machine. We consider that the setup times verify the triangle inequality, i.e., $s_{uv} + s_{vw} \geq s_{uw}$ holds for any operations u, v and w requiring the same machine, as it usually happens in real scenarios.

Furthermore, to add flexibility to the problem, an operation θ_{ij} can be executed by one machine out of a set $M_{ij} \subseteq M$ of given machines. The processing time for operation θ_{ij} on machine $k \in M_{ij}$ is $p_{ijk} \in \mathbb{N}$. Notice that the processing time of an operation may be different in each machine and that a machine may process several operations of the same job. To simplify notation, in the remaining of the paper p_v denote the processing time of an operation v in its current machine assignment.

Let Ω denote the set of operations. A solution may be viewed as a feasible assignment of machines to operations and a processing order of the operations on the machines (i.e., a machine sequence). So, given a machine assignment, a solution may be represented by a total ordering of the operations, σ, compatible with the jobs and the machines sequences. For an operation $v \in \Omega$ let PJ_v and SJ_v denote the operations just before and after v in the job sequence and PM_v and SM_v the operations right before and after v in the machine sequence in a solution σ. The starting and completion times of v, denoted t_v and c_v respectively, can be calculated as $t_v = \max(c_{PJ_v}, c_{PM_v} + s_{PM_v v})$ and $c_v = t_v + p_v$. The objective is to find a solution σ that minimizes the makespan, i.e., the completion time of the last operation, denoted as $C_{max}(\sigma) = \max_{v \in \Omega} c_v$.

2.1 Solution graph

We define a solution graph model which is adapted from (Vela, Varela, and González 2010) to deal with machine flexibility. In accordance with this model, a feasible operation processing order σ can be represented by an acyclic directed graph G_σ where each node represents either an operation of the problem or one of the dummy nodes *start* and *end*, which are fictitious operations with processing time 0. In G_σ, there are *conjunctive arcs* representing job processing

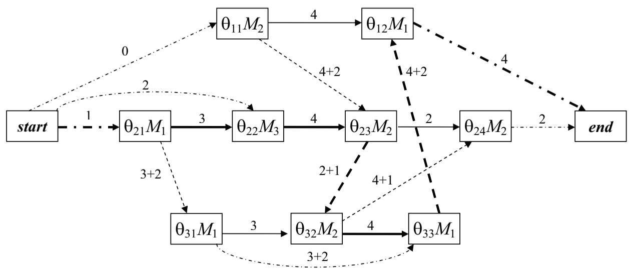

Figure 1: A feasible schedule to a problem with 3 jobs and 3 machines represented by a solution graph. Bold-face arcs show a critical path whose length, i.e., the makespan, is 25.

orders and *disjunctive arcs* representing machine processing orders. Each disjunctive arc (v, w) is weighted with $p_v + s_{vw}$ and each conjunctive arc (u, w) is weighted with p_u. If w is the first operation in the machine processing order, there is an arc $(start, w)$ in G_σ with weight s_{0w} and if w is the last operation, in both the job and machine sequences, there is an arc (w, end) with weight p_w. Figure 1 shows a solution graph.

The makespan of the schedule is the cost of a critical path in G_σ, i.e., a directed path from node $start$ to node end having maximum cost. Bold-face arcs in Figure 1 represent a critical path. Nodes and arcs in a critical path are also termed critical. We define a critical block as a maximal subsequence of consecutive operations in a critical path requiring the same machine such that two consecutive operations of the block do not belong to the same job. As we will see in Section 3, this point is important as the order of two operations in the same job cannot be reversed. Most neighborhood structures proposed for job shop problems rely on exchanging the processing order of operations in critical blocks (Dell' Amico and Trubian 1993; Van Laarhoven, Aarts, and Lenstra 1992).

To formalize the description of the neighborhood structures, we introduce the concepts of head and tail of an operation v, denoted r_v and q_v respectively. Heads and tails are calculated as follows:

$$r_{start} = q_{end} = 0$$
$$r_v = \max(r_{PJ_v} + p_{PJ_v}, r_{PM_v} + p_{PM_v} + s_{PM_v v})$$
$$r_{end} = \max_{v \in PJ_{end} \cap PM_{end}} \{r_v + p_v\}$$

$$q_v = \max(q_{SJ_v} + p_{SJ_v}, q_{SM_v} + p_{SM_v} + s_{vSM_v})$$
$$q_{start} = \max_{v \in SM_{start}} \{q_v + p_v + s_{0v}\}$$

Here, we abuse notation slightly, so SM_{start} (resp. PM_{end}) denotes the set formed by the first (resp. last) operation processed in each of the m machines and PJ_{end} denotes the set formed by the last operation processed in each of the n jobs. A node v is critical if and only if $C_{max} = r_v + p_v + q_v$.

3 Neighborhood structure

We propose here a neighborhood structure for the SDST-FJSP termed N_1^{SF}. This structure considers two types of moves. Firstly, reversals of single critical arcs, which are analogous to the moves of the structure N_1^S defined in (Vela, Varela, and González 2010) for the FJSP. Furthermore, we consider moves which have to do with machine assignment to operations. Abusing notation, we use the notation $N_1^{SF} = N_1^S \cup N_1^F$ to indicate that the new structure is the union of these two subsets of moves.

3.1 N_1^S structure

For the sequencing subproblem, we consider single moves, i.e., reversing the processing order of two consecutive operations. We use a filtering mechanism based on the results below, which allow the algorithm to discard unfeasible and a number of non-improving neighbors. By doing this, we get a neighborhood of reasonable size while augmenting the chance of obtaining improving neighbors.

The next result establishes a sufficient condition for non-improvement when a single arc is reversed in a solution.

Proposition 1 *Let σ be a schedule and (v, w) a disjunctive arc which is not in a critical block. Then, if the setup times fulfill the triangular inequality, reversing the arc (v, w) does not produce any improvement even if the resulting schedule σ' is feasible.*

Then, as we consider that the setup times fulfill the triangular inequality, we will only consider reversing critical arcs in order to obtain improving schedules. However, reversing some of the critical arcs cannot produce improving schedules as it is established in the following result.

Proposition 2 *Let σ be a schedule and (v, w) an arc inside a critical block B, i.e., PM_v and SM_w belong to B. Even if the schedule σ' obtained from σ by reversing the arc (v, w) is feasible, σ' does not improve σ if the following condition holds*

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

where $x = PM_v$ and $y = SM_w$ in schedule σ.

Regarding feasibility, the following result guarantees that the resulting schedule after reversing the arc (v, w) is feasible.

Proposition 3 *Let σ be a schedule and (v, w) an arc in a critical block. A sufficient condition for an alternative path between v and w not to exist is that*

$$r_{PJ_w} < r_{SJ_v} + p_{SJ_v} + C \qquad (2)$$

whre $C = s_{zPJ_w}$ if $SM_z = PJ_w$, and $C = min\{p_{SJ_z}, s_{zSM_z} + p_{SM_z}\}$ otherwise, being $z = SJ_v$.

Therefore, in N_1^S, we only consider reversing arcs (v, w) in a critical block which fulfill the condition (2), provided that condition (1) does not hold.

Notice that, when one of the neighbors is finally selected, we need to reconstruct the total ordering σ of the operations. If the neighbor was created by N_1^S reversing an arc (v, w) then we have to reconstruct σ starting from the position of operation v and finishing in the position of w (a similar reconstruction is detailed in (Mattfeld 1995) for the classic JSP).

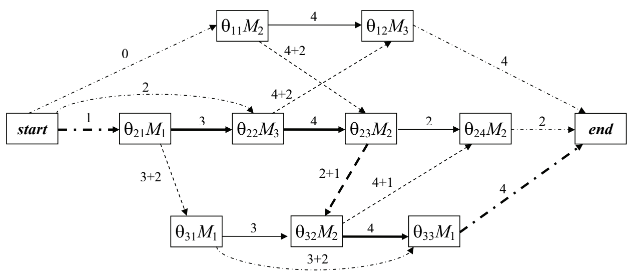

Figure 2: A neighbor of the schedule of Figure 1 created with N_1^F. The makespan is reduced from 25 to 19.

3.2 N_1^F structure

Changing the machine of one operation may give rise to an improved schedule. This possibility was already exploited in (Mastrolilli and Gambardella 2000) where the authors consider a set of moves, called k-insertions, in which an operation v is assigned to a new machine $k \in M_v$ and then they look for the optimal position for v in the machine sequence of k. However, that procedure is time consuming, so we propose here a simpler approach which works as follows.

Let σ be a total ordering of the operations in a schedule and let v be an operation in a critical path of G_σ, an element of N_1^F is obtained by assigning a new machine $k \in M_v$ to the operation v and then selecting a position for v in the machine sequence of k so that the topological order given by σ remains identical after the move. Clearly, this neighbor is feasible and the method is very efficient and not time consuming. The number of neighbors depends of the number of machines suitable for each operation on the critical path.

As an example, notice that M_3 in Figure 1 only processes the operation θ_{22}. If M_3 can process θ_{12}, N_1^F switches the machine assignation of the critical task θ_{12} from M_1 to M_3 to create a new neighbor. As θ_{12} is after θ_{22} in the topological order, then N_1^F inserts θ_{12} after θ_{22} in the new machine sequence of M_3. The result is the schedule of Figure 2, whose makespan is better than that of the original schedule.

3.3 Makespan estimate

Computing the actual makespan of a neighbor is computationally expensive, since it requires recalculating the head of all operations after v and the tail of all operations before w, when the arc (v, w) is reversed. On the other hand, it requires recalculating the head of all operations after v and the tail of all operations before v when v is assigned to a different machine. So, as it is usual, we propose using estimation procedures instead.

For N_1^S, we borrow the estimation used in (Vela, Varela, and González 2010), which is based on the $lpath$ procedure for the classical JSP from (Taillard 1993). To calculate this estimate, after reversing the arc (v, w) in a schedule σ to obtain σ', if $x = PM_v$ and $y = SM_w$ before the move, the heads and tails for operations v and w in σ' are estimated as follows:

$$\begin{aligned} r'_w &= \max\{r_{PJ_w} + p_{PJ_w}, r_x + p_x + s_{xw}\} \\ r'_v &= \max\{r_{PJ_v} + p_{PJ_v}, r'_w + p_w + s_{wv}\} \\ q'_v &= \max\{q_{SJ_v} + p_{SJ_v}, q_y + p_y + s_{vy}\} \\ q'_w &= \max\{q_{SJ_w} + p_{SJ_w}, q'_v + p_v + s_{wv}\} \end{aligned}$$

Given this, the makespan of σ' can be estimated as the maximum length of the longest paths from node $start$ to node end through nodes v and w, namely $Est(C_{max}(\sigma')) = \max\{r'_w + p_w + q'_w, r'_v + p_v + q'_v\}$. This procedure returns a lower bound of the makespan of σ'.

Regarding N_1^F, if we change the machine assignation of an operation v, a fast estimation can be obtained calculating the longest path through v in the new schedule σ'. To this end, we estimate the head and tail of v in σ' as follows:

$$\begin{aligned} r'_v &= \max\{r_{PJ_v} + p_{PJ_v}, r_{PM_v} + p_{PM_v} + s_{PM_v v}\} \\ q'_v &= \max\{q_{SJ_v} + p_{SJ_v}, q_{SM_v} + p_{SM_v} + s_{vSM_v}\} \end{aligned}$$

The makespan of σ' can be estimated as $Est(C_{max}(\sigma')) = r'_v + p_v + q'_v$ (notice that p_v may change from σ to σ'). This is also a lower bound of the makespan of σ'.

In order to evaluate the accuracy of the estimates, we estimated and evaluated the actual makespan of about 100 million neighbors for instances with different sizes. With regard to N_1^S, we observed that the estimate coincided with the exact value of the makespan in 88.9% of the neighbors. In the remaining 11.1% of the neighbors, the estimate was in average 2.30% lower than the actual makespan. For N_1^F, the estimate coincided in 94.0% of the neighbors, and in the remaining 6.0% the estimate was in average 3.16% lower than the actual makespan. Therefore, these estimates are really efficient and appropriate.

4 Memetic algorithm

In this section, we describe the main characteristics of the memetic algorithm used. Firstly, the genetic algorithm and then the tabu search which is applied to every chromosome generated by the genetic algorithm.

4.1 Genetic Algorithm

We use a conventional genetic algorithm where the initial population is generated at random. Then, the algorithm iterates over a number of steps or generations. In each iteration, a new generation is built from the previous one by applying selection, recombination and replacement operators.

In the selection phase all chromosomes are grouped into pairs, and then each one of these pairs is mated to obtain two offspring. Tabu search is applied to both offspring, and finally the replacement is carried out as a tournament selection from each pair of parents and their two offspring. After tabu search, a chromosome is rebuilt from the improved schedule obtained, so its characteristics can be transferred to subsequent offsprings. This effect of the evaluation function is known as Lamarckian evolution.

The coding schema is based on the two-vector representation, which is widely used in the flexible job-shop problem (see for example (Gao, Sun, and Gen 2008)). In this representation, each chromosome has one vector with the task sequence and another one with the machine assignment.

The task sequence vector is based on permutations with repetition, as proposed in (Bierwirth 1995) for the JSP. It is a permutation of the set of operations, each being represented by its job number. For example, if we have a problem with 3 jobs: $J_1 = \{\theta_{11}, \theta_{12}\}, J_2 = \{\theta_{21}, \theta_{22}, \theta_{23}, \theta_{24}\}, J_3 = \{\theta_{31}, \theta_{32}, \theta_{33}\}$, then the sequence (2 1 2 3 2 3 3 2 1) is a valid vector that represents the topological order $\{\theta_{21}, \theta_{11}, \theta_{22}, \theta_{31}, \theta_{23}, \theta_{32}, \theta_{33}, \theta_{24}, \theta_{12}\}$. With this encoding, every permutation produces a feasible processing order.

The machine assignment vector has the machine number that uses the task located in the same position in the task sequence vector. For example, if we consider the sequence vector above, then the machine vector (1 2 3 1 2 2 1 2 1), indicates that the tasks θ_{21}, θ_{31}, θ_{33} and θ_{12} use the machine 1, the tasks θ_{11}, θ_{23}, θ_{32} and θ_{24} use the machine 2, and only the task θ_{22} uses the machine 3.

For chromosome mating the genetic algorithm uses an extension of the Job Order Crossover (JOX) described in (Bierwirth 1995) for the classical JSP. Given two parents, JOX selects a random subset of jobs and copies their genes to one offspring in the same positions as in the first parent, then the remaining genes are taken from the second parent so that they maintain their relative ordering. For creating another offspring the parents change their roles. In extending this operator to the flexible case, we need to consider also the machine assignment vector. We propose choosing for every task the assignation it has in the parent it comes from. We clarify how this extended JOX operator works by means of an example. Let us consider the following two parents

Parent 1 Sequence: (2 1 2 3 2 3 3 2 1)
Parent 1 Assignment: (1 2 3 1 2 2 1 2 1)
Parent 2 Sequence: (1 3 2 2 1 3 2 2 3)
Parent 2 Assignment: (3 2 3 1 3 2 1 3 3)
If the selected subset of jobs just includes the job 2, then
Offspring 1 Sequence: (2 1 2 3 2 1 3 2 3)
Offspring 1 Assignment: (1 3 3 2 2 3 2 2 3)
Offspring 2 Sequence: (1 3 2 2 3 3 2 2 1)
Offspring 2 Assignment: (2 1 3 1 2 1 1 3 1)

The operator JOX might swap any two operations requiring the same machine; this is an implicit mutation effect. For this reason, we have not used any explicit mutation operator. Therefore, parameter setting in the experimental study is considerably simplified, as crossover probability is set to 1 and mutation probability need not be specified. With this setting, we have obtained results similar to those obtained with a lower crossover probability and a low probability of applying mutation operators. Some authors, for example in (Essafi, Mati, and Dauzère-Pérès 2008) or (González et al. 2012) have already noticed that a mutation operator does not play a relevant role in a memetic algorithm.

To build schedules we have used a simple decoding algorithm: the operations are scheduled in exactly the same order as they appear in the chromosome sequence σ. In other

Table 1: Summary of results in the SDST-FJSP: SDST-HUdata benchmark

Instance	Size	Flex.	LB	IFS	GA	TS	GA+TS	T(s.)
la01	10×5	1.15	609	726	801 (817)	**721(*)** (724)	**721(*)** (724)	6
la02	10×5	1.15	655	749	847 (870)	**737(*)** (738)	**737(*)** (737)	7
la03	10×5	1.15	550	**652**	760 (789)	**652** (652)	**652** (652)	7
la04	10×5	1.15	568	**673**	770 (790)	**673** (678)	**673** (675)	9
la05	10×5	1.15	503	603	679 (685)	**602(*)** (602)	**602(*)** (602)	8
la06	15×5	1.15	833	**950**	1147 (1165)	956 (961)	953 (957)	12
la07	15×5	1.15	762	916	1123 (1150)	912(*) (917)	**905(*)** (911)	18
la08	15×5	1.15	845	948	1167 (1186)	**940(*)** (951)	**940(*)** (941)	15
la09	15×5	1.15	878	1002	1183 (1210)	1002 (1007)	**989(*)** (995)	22
la10	15×5	1.15	866	977	1127 (1156)	**956(*)** (960)	**956(*)** (956)	29
la11	20×5	1.15	1087	1256	1577 (1600)	1265 (1273)	**1244(*)** (1254)	33
la12	20×5	1.15	960	**1082**	1365 (1406)	1105 (1119)	1098 (1107)	26
la13	20×5	1.15	1053	1215	1473 (1513)	1210(*) (1223)	**1205(*)** (1212)	24
la14	20×5	1.15	1123	1285	1549 (1561)	1267(*) (1277)	**1257(*)** (1263)	27
la15	20×5	1.15	1111	1291	1649 (1718)	1284(*) (1297)	**1275(*)** (1282)	29
la16	10×10	1.15	892	**1007**	1256 (1269)	**1007** (1007)	**1007** (1007)	12
la17	10×10	1.15	707	858	1007 (1059)	**851(*)** (851)	**851(*)** (851)	12
la18	10×10	1.15	842	**985**	1146 (1184)	**985** (988)	**985** (992)	10
la19	10×10	1.15	796	956	1166 (1197)	**951(*)** (955)	**951(*)** (951)	16
la20	10×10	1.15	857	**997**	1194 (1228)	**997** (997)	**997** (997)	12
MRE				16.29	38.81 (42.27)	15.93 (16.49)	15.55 (15.92)	
#best				7	0	12	18	

Values in **bold** are best known solutions, (*) improves previous best known solution.

words, we produce a semiactive schedule, which means that no operation can start earlier without altering the operation sequence for a given machine assignment.

4.2 Tabu Search

Tabu search (TS) is an advanced local search technique, proposed in (Glover 1989a) and (Glover 1989b), which can escape from local optima by selecting non-improving neighbors. To avoid revisiting recently visited solutions and explore new promising regions of the search space, it maintains a tabu list with a set of moves which are not allowed when generating the new neighborhood. TS has a solid record of good empirical performance in problem solving. For example, the $i-TSAB$ algorithm from (Nowicki and Smutnicki 2005) is one of the best approaches for the JSP. TS is often used in combination with other metaheuristics.

The general TS scheme used in this paper is similar to that proposed in (Dell' Amico and Trubian 1993). In the first step the initial solution (provided by the genetic algorithm, as we have seen) is evaluated. It then iterates over a number of steps. At each iteration, a new solution is selected from the neighborhood of the current solution using the estimated makespan as selection criterion. A neighbor is tabu if it is generated by reversing a tabu arc, unless its estimated makespan is better than that of the current best solution. Additionally, we use the dynamic length schema for the tabu list and the cycle checking mechanism as it is proposed in (Dell' Amico and Trubian 1993). TS finishes after a number of iterations without improvement, returning the best solution reached so far.

5 Experimental study

We have conducted an experimental study across benchmarks of common use for both problems, the SDST-FJSP and the FJSP. In both cases our memetic algorithm (GA+TS), was given a population of 100 chromosomes and stopped after 20 generations without improving the best solution of the population. Also, the stopping criterion for tabu search is set to 400 iterations without improvement. We have implemented our method in C++ on a PC with Intel Core 2 Duo at 2.66 GHz and 2 Gb RAM.

We also show the results produced by the genetic algorithm (GA) and the TS approaches separately, to compare each of them with the hybridized approach. GA was given a population of 100 chromosomes and the stopping criterion is the run time used by GA+TS (i.e., it is not the maximum number of generations). For running TS alone we have opted to set the same stopping criterion as in GA+TS (400 iterations without improvement), and to launch TS starting from random schedules as many times as possible in the run time used by GA+TS. We have tried several different configurations for running GA and TS alone, with similar or worse results than those described here.

5.1 Comparison with the state-of-the-art in the SDST-FJSP

As we have pointed, there are few papers that tackle the SDST-FJSP. To our knowledge, the most representative approach to this problem is the iterative flattening search (IFS) proposed in (Oddi et al. 2011). So we choose this method to compare with our proposal. We consider the same bench-

Table 2: Summary of results in the FJSP: DP Benchmark

Instance	Size	Flex.	LB	TS	hGA	CDDS	GA+TS	T(s.)
01a	10×5	1.13	2505	2518 (2528)	2518 (2518)	2518 (2525)	**2505(*)** (2511)	74
02a	10×5	1.69	2228	**2231** (2234)	**2231** (2231)	**2231** (2235)	2232 (2234)	120
03a	10×5	2.56	2228	**2229** (2230)	**2229** (2229)	**2229** (2232)	**2229** (2230)	143
04a	10×5	1.13	2503	**2503** (2516)	2515 (2518)	**2503** (2510)	**2503** (2504)	72
05a	10×5	1.69	2189	**2216** (2220)	2217 (2218)	**2216** (2218)	2219 (2221)	123
06a	10×5	2.56	2162	2203 (2206)	**2196** (2198)	**2196** (2203)	2200 (2204)	157
07a	15×8	1.24	2187	2283 (2298)	2307 (2310)	2283 (2296)	**2266(*)** (2286)	201
08a	15×8	2.42	2061	**2069** (2071)	2073 (2076)	**2069** (2069)	2072 (2075)	197
09a	15×8	4.03	2061	**2066** (2067)	**2066** (2067)	**2066** (2067)	**2066** (2067)	291
10a	15×8	1.24	2178	2291 (2306)	2315 (2315)	2291 (2303)	**2267(*)** (2273)	240
11a	15×8	2.42	2017	**2063** (2066)	2071 (2072)	**2063** (2072)	2068 (2071)	222
12a	15×8	4.03	1969	2034 (2038)	**2030** (2031)	2031 (2034)	2037 (2041)	266
13a	20×10	1.34	2161	2260 (2266)	**2257** (2260)	**2257** (2260)	2271 (2276)	241
14a	20×10	2.99	2161	**2167** (2168)	**2167** (2168)	**2167** (2179)	2169 (2171)	340
15a	20×10	5.02	2161	2167 (2167)	**2165** (2165)	**2165** (2170)	2166 (2166)	470
16a	20×10	1.34	2148	**2255** (2259)	2256 (2258)	2256 (2258)	2266 (2271)	253
17a	20×10	2.99	2088	2141 (2144)	**2140** (2142)	**2140** (2146)	2147 (2150)	333
18a	20×10	5.02	2057	2137 (2140)	**2127** (2131)	**2127** (2132)	2138 (2141)	488
MRE				2.01 (2.24)	2.12 (2.19)	1.94 (2.19)	1.99 (2.17)	
#best				9	10	13	6	

Values in **bold** are best known solutions, (*) improves previous best known solution.

mark used in that paper, which is denoted SDST-HUdata. It consists of 20 instances derived from the first 20 instances of the data subset of the FJSP benchmark proposed in (Hurink, Jurisch, and Thole 1994). Each instance was created by adding to the original instance one setup time matrix st^r for each machine r. The same setup time matrix was added for each machine in all benchmark instances. Each matrix has size $n \times n$, and the value st^r_{ij} indicates the setup time needed to reconfigure the machine r when switches from job i to job j. These setup times are sequence dependent and they fulfill the triangle inequality.

IFS is implemented in Java and run on a AMD Phenom II X4 Quad 3.5 Ghz under Linux Ubuntu 10.4.1, with a maximum CPU time limit set to 800 seconds for all runs. We are considering here the best makespan reported in (Oddi et al. 2011) for each instance, regardless of the configuration used.

Table 1 shows the results of the experiments in the SDST-HUdata benchmark. In particular we indicate for each instance the name, the size ($n \times m$), the flexibility (i.e. the average number of available machines per operation) and a lower bound LB. The lower bounds are those reported in (Mastrolilli and Gambardella 2000) for the original instances without setups, therefore they are probably far from the optimal solutions. For IFS we indicate the best results reported in (Oddi et al. 2011), and for GA, TS and GA+TS we indicate the best and average makespan in 10 runs for each instance. We also show the runtime in seconds of a single run of our algorithms. Additionally, we report the MRE (Mean Relative Error) for each method, calculated as follows: $MRE = (C_{max} - LB)/LB \times 100$. Finally, in the bottom line we indicate the number of instances for which a method obtains the best known solution (#best). We mark in bold the best known solutions, and we mark with a "(*)" when a method improves the previous best known solution.

We notice that GA alone obtains very poor results, with a MRE of 38.81% for the best solutions. TS alone performs much better than GA, with a MRE of 15.93% for the best solutions, and was able to reach the best known solution in 12 of the 20 instances. However, the hybridized approach obtains even better results. In fact, GA+TS obtains a better average makespan than TS in 13 instances, the same in 6 instances and a worse average in only 1 instance. This shows the good synergy between the two metaheuristics.

Overall, compared to IFS, GA+TS establishes new best solutions for 13 instances, reaches the same best known solution for 5 instances and for the instances la06 and la12 the solution reached is worse than the current best known solution. Regarding the average makespan, it is better than the best solution obtained by IFS in 13 instances, it is the equal in 3 instances and it is worse in 4 instances. IFS achieved a MRE of 16.29%, while GA+TS achieved a MRE of 15.55% for the best solutions and 15.92% for the average values. Additionally, the CPU time of GA+TS (between 6 and 33 seconds per run depending on the instance) is lower than that of IFS (800 seconds per run). However CPU times are not directly comparable due to the differences in programming languages, operating systems and target machines. In conclusion, GA+TS is better than GA and TS alone, and it is quite competitive with IFS.

5.2 Comparison with the state-of-the-art in the FJSP

In the FJSP it is easier to compare with the state-of-the-art, because of the number of existing works. We con-

Table 3: Summary of results in the FJSP: BC Benchmark

Instance	Size	Flex.	LB	TS	hGA	CDDS	GA+TS	T(s.)
mt10c1	10×11	1.10	655	928 (928)	**927** (927)	928 (929)	**927** (927)	14
mt10cc	10×12	1.20	655	910 (910)	910 (910)	910 (911)	**908(*)** (909)	14
mt10x	10×11	1.10	655	**918** (918)	**918** (918)	**918** (918)	**918** (922)	18
mt10xx	10×12	1.20	655	**918** (918)	**918** (918)	**918** (918)	**918** (918)	16
mt10xxx	10×13	1.30	655	**918** (918)	**918** (918)	**918** (918)	**918** (918)	19
mt10xy	10×12	1.20	655	906 (906)	**905** (905)	906 (906)	**905** (905)	16
mt10xyz	10×13	1.30	655	**847** (850)	849 (849)	849 (851)	849 (850)	21
setb4c9	15×11	1.10	857	919 (919)	**914** (914)	919 (919)	**914** (914)	22
setb4cc	15×12	1.20	857	909 (912)	914 (914)	909 (911)	**907(*)** (907)	22
setb4x	15×11	1.10	846	**925** (925)	**925** (931)	**925** (925)	**925** (925)	18
setb4xx	15×12	1.20	846	**925** (926)	**925** (925)	**925** (925)	**925** (925)	19
setb4xxx	15×13	1.30	846	**925** (925)	**925** (925)	**925** (925)	**925** (925)	20
setb4xy	15×12	1.20	845	916 (916)	916 (916)	916 (916)	**910(*)** (910)	25
setb4xyz	15×13	1.30	838	**905** (908)	**905** (905)	**905** (907)	**905** (905)	19
seti5c12	15×16	1.07	1027	1174 (1174)	1175 (1175)	1174 (1175)	**1171(*)** (1173)	41
seti5cc	15×17	1.13	955	**1136** (1136)	1138 (1138)	**1136** (1137)	**1136** (1137)	40
seti5x	15×16	1.07	955	1201 (1204)	1204 (1204)	1201 (1202)	**1199(*)** (1200)	43
seti5xx	15×17	1.13	955	1199 (1201)	1202 (1203)	1199 (1199)	**1197(*)** (1198)	38
seti5xxx	15×18	1.20	955	**1197** (1198)	1204 (1204)	**1197** (1198)	**1197** (1197)	40
seti5xy	15×17	1.13	955	**1136** (1136)	**1136** (1137)	**1136** (1138)	**1136** (1137)	39
seti5xyz	15×18	1.20	955	**1125** (1127)	1126 (1126)	**1125** (1125)	1127 (1128)	41
MRE				22.53 (22.63)	22.61 (22.66)	22.54 (22.60)	22.42 (22.49)	
#best				12	11	11	19	

Values in **bold** are best known solutions, (*) improves previous best known solution.

sider several sets of problem instances: the DP benchmark proposed in (Dauzère-Pérès and Paulli 1997) with 18 instances, the BC benchmark proposed in (Barnes and Chambers 1996) with 21 instances, and the BR benchmark proposed in (Brandimarte 1993) with 10 instances.

We are comparing GA+TS with the tabu search (TS) of (Mastrolilli and Gambardella 2000), the hybrid genetic algorithm (hGA) of (Gao, Sun, and Gen 2008) and the climbing depth-bounded discrepancy search (CDDS) of (Hmida et al. 2010). These three methods are, as far as we know, the best existing approaches.

TS was coded in C++ on a 266 MHz Pentium. They execute 5 runs per instance and they limit the maximum number of iterations between 100000 and 500000 depending on the instance. With this configuration they report run times between 28 and 150 seconds in the DP benchmark, between 1 and 24 seconds in the BC benchmark, and between 0.01 and 8 seconds in the BR benchmark.

hGA was implemented in Delphi on a 3.0 GHz Pentium. Depending on the complexity of the problems, the population size of hGA ranges from 300 to 3000, and the number of generations is limited to 200. They also execute 5 runs per instance, and with the described configuration they report run times between 96 and 670 seconds in the DP benchmark, between 10 and 72 seconds in the BC benchmark, and between 1 and 20 seconds in the BR benchmark.

CDDS was coded in C on an Intel Core 2 Duo 2.9 GHz PC with 2GB of RAM. It is a deterministic algorithm, however in the paper are reported the results of 4 runs per instance, one for each of the neighborhood structures they propose, therefore we report the best and average solutions for the method. The authors set the maximum CPU time to 15 seconds for all test instances except for DP benchmark, in which the maximum CPU time is set to 200 seconds.

GA+TS was run 10 times for each instance. The run time of the algorithm is in direct ratio with the size and flexibility of the instance. The CPU times range from 72 to 488 seconds in the DP benchmark, from 14 to 43 seconds in the BC benchmark, and from 6 to 112 seconds in the BR benchmark. Therefore, the run times are similar to that of the other methods, although they are not directly comparable due to the differences in languages and target machines. However, we have seen that in easy instances the best solution is found in the very first generations, therefore, for many of these instances GA+TS requires a much smaller CPU time to reach the same solution.

Tables 2, 3 and 4 show the results of the experiments in the DP benchmark, BC benchmark and BR benchmark, respectively. As we did for Table 1, we indicate for each instance the name, size, flexibility, and the lower bound reported in (Mastrolilli and Gambardella 2000). Then, for each method we report the best and average makespan. We also indicate the runtime of a single run of GA+TS. And finally, the MRE for each method and the number of instances for which a method reaches the best known solution.

In the DP benchmark GA+TS improves the previous best known solution in 3 of the 18 instances (01a 07a 10a). It is remarkable that we prove the optimality of the solution 2505 for instance 01a, as this is also its lower bound. Moreover, the MRE of the average makespan of GA+TS is the best of

Table 4: Summary of results in the FJSP: BR Benchmark

Instance	Size	Flex.	LB	TS	hGA	CDDS	GA+TS	T(s.)
Mk01	10×6	2.09	36	**40** (40)	**40** (40)	**40** (40)	**40** (40)	6
Mk02	10×6	4.10	24	**26** (26)	**26** (26)	**26** (26)	**26** (26)	13
Mk03	15×8	3.01	204	**204** (204)	**204** (204)	**204** (204)	**204** (204)	9
Mk04	15×8	1.91	48	**60** (60)	**60** (60)	**60** (60)	**60** (60)	20
Mk05	15×4	1.71	168	173 (173)	**172** (172)	173 (174)	**172** (172)	17
Mk06	10×15	3.27	33	**58** (58)	**58** (58)	**58** (59)	**58** (58)	54
Mk07	20×5	2.83	133	144 (147)	**139** (139)	**139** (139)	**139** (139)	40
Mk08	20×10	1.43	523	**523** (523)	**523** (523)	**523** (523)	**523** (523)	14
Mk09	20×10	2.53	299	**307** (307)	**307** (307)	**307** (307)	**307** (307)	26
Mk10	20×15	2.98	165	198 (199)	**197** (197)	**197** (198)	199 (200)	112
MRE				15.41 (15.83)	14.92 (14.92)	14.98 (15.36)	15.04 (15.13)	
#best				7	10	9	9	

Values in **bold** are best known solutions.

the four algorithms considered (2.17% versus 2.19%, 2.19% and 2.24%). The MRE of the best makespan is the second best of the four algorithms, although in this case we have to be aware that our algorithm has some advantage since we launched it more times for each instance.

In the BC benchmark we are able to improve the previous best known solution in 6 of the 21 instances (mt10cc, setb4cc, setb4xy, seti5c12, seti5x and seti5xx). Additionally, the MRE of both the best and average makespan of GA+TS is the best of the four algorithms considered. In particular, considering the best makespan our MRE is 22.42% (versus 22.53%, 22.54% and 22.61%), and considering the average makespan our MRE is 22.49% (versus 22.60%, 22.63% and 22.66%). Moreover, the number of instances for which we obtain the best known solution is the best of the four methods: 19 of the 21 instances.

In the BR benchmark we obtain the best known solution in 9 of the 10 instances; only hGA is able to improve that number. Regarding the MRE, GA+TS is second best considering the average makespan and third best considering the best makespan. We can conclude that BR benchmark is generally easy, as for most instances all four methods reached the best known solution in every run. In many of these instances GA+TS reached the best known solution in the first generation of each run.

Overall, GA+TS shows a lower efficiency in the largest and most flexible instances, compared to the other algorithms. In our opinion this is due to the fact that GA+TS needs more run time to improve the obtained solution in these instances. Additionally, regarding run times in the FJSP benchmarks we have to notice that our algorithm is at disadvantage, because it does all the necessary setup calculations even when the setup times are zero, as it occurs in these instances.

In summary, we can conclude that GA+TS is competitive with the state-of-the-art and was able to obtain new upper bounds for 9 of the 49 instances considered.

6 Conclusions

We have considered the flexible job shop scheduling problem with sequence-dependent setup times, where the objective is to minimize the makespan. We have proposed a neighborhood structure termed N_1^{SF} which is the union of two structures: N_1^S, designed to modify the sequencing of the tasks, and N_1^F, designed to modify the machine assignations. This structure has then been used in a tabu search algorithm, which is embedded in a genetic algorithm framework. We have defined methods for estimating the makespan of both neighborhoods and empirically shown that they are very accurate. In the experimental study we have shown that the hybridized approach (GA+TS) is better than each method alone. We have also compared our approach against state-of-the-art algorithms, both in FJSP and SDST-FJSP benchmarks. In the SDST-FJSP we have compared GA+TS with the algorithm proposed in (Oddi et al. 2011) across the 20 instances described in that paper. For the FJSP we have considered three sets of instances, comparing GA+TS with the methods proposed in (Gao, Sun, and Gen 2008), (Mastrolilli and Gambardella 2000) and (Hmida et al. 2010). Our proposal compares favorably to state-of-the-art methods in both problems considered, and we are able to improve the best known solution in 9 of the 49 FJSP instances and in 13 of the 20 SDST-FJSP instances.

As future work we plan to experiment across the FJSP benchmark proposed in (Hurink, Jurisch, and Thole 1994), and also across the SDST-FJSP benchmark proposed in (Saidi-Mehrabad and Fattahi 2007). We also plan to define a new SDST-FJSP benchmark with bigger instances and with more flexibility than the ones proposed in (Oddi et al. 2011). We shall also consider different crossover operators and other metaheuristics like scatter search with path relinking. Finally, we plan to extend our approach to other variants of scheduling problems which are even closer to real environments, for instance, problems with uncertain durations, or problems considering alternative objective functions such as weighted tardiness.

Acknowledgements

We would like to thank Angelo Oddi and Riccardo Rasconi for providing us with the SDST-FJSP instances. All authors are supported by Grant MEC-FEDER TIN2010-20976-C02-02.

References

Barnes, J., and Chambers, J. 1996. Flexible job shop scheduling by tabu search. *Technical Report Series: ORP96-09, Graduate program in operations research and industrial engineering. The University of Texas at Austin*.

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

Brandimarte, P. 1993. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research* 41:157–183.

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.

Dauzère-Pérès, S., and Paulli, J. 1997. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research* 70(3):281–306.

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

Essafi, I.; Mati, Y.; and Dauzère-Pérès, S. 2008. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers and Operations Research* 35:2599–2616.

Gao, J.; Sun, L.; and Gen, M. 2008. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers and Operations Research* 35:2892–2907.

Glover, F. 1989a. Tabu search–part I. *ORSA Journal on Computing* 1(3):190–206.

Glover, F. 1989b. Tabu search–part II. *ORSA Journal on Computing* 2(1):4–32.

González, M. A.; González-Rodríguez, I.; Vela, C.; and Varela, R. 2012. An efficient hybrid evolutionary algorithm for scheduling with setup times and weighted tardiness minimization. *Soft Computing* 16(12):2097–2113.

González, M. A.; Vela, C.; and Varela, R. 2012. A competent memetic algorithm for complex scheduling. *Natural Computing* 11:151–160.

Hmida, A.; Haouari, M.; Huguet, M.; and Lopez, P. 2010. Discrepancy search for the flexible job shop scheduling problem. *Computers and Operations Research* 37:2192–2201.

Hurink, E.; Jurisch, B.; and Thole, M. 1994. Tabu search for the job shop scheduling problem with multi-purpose machine. *Operations Research Spektrum* 15:205–215.

Mastrolilli, M., and Gambardella, L. 2000. Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling* 3(1):3–20.

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

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

Oddi, A.; Rasconi, R.; Cesta, A.; and Smith, S. 2011. Applying iterative flattening search to the job shop scheduling problem with alternative resources and sequence dependent setup times. In *COPLAS 2011 Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*.

Saidi-Mehrabad, M., and Fattahi, P. 2007. Flexible job shop scheduling with tabu search algorithms. *Int J Adv Manuf Technol* 32:563–570.

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

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

Vela, C. R.; Varela, R.; and González, M. A. 2010. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics* 16:139–165.