Scatter Search and Path Relinking for the Job Shop Scheduling Problem with Sequence Dependent and non-Anticipatory Setup Times

Miguel A. González[1], Camino R. Vela[1], Ramiro Varela[1], and Inés González-Rodríguez[2]

[1] Department of Computing, Artificial Intelligence Center
University of Oviedo, Spain
mig@uniovi.es , crvela@uniovi.es , ramiro@uniovi.es

[2] Department of Mathematics, Statistics and Computing, University of Cantabria, Spain ines.gonzalez@unican.es

Abstract. We tackle the job shop scheduling problem with sequence-dependent non-anticipatory setup times where the objective is to minimize the makespan. To solve it, we design a scatter search algorithm which incorporates path relinking and tabu search. The performance of this algorithm relies on a new neighborhood structure proposed in this paper based on a graph model representing the non-anticipatory characteristic of setup times. We have conducted an experimental study across usual benchmarks to analyze our algorithm and to compare it with the state-of-the-art. In these experiments, our approach establishes new best solutions for all the instances.

1 Introduction

In the job shop scheduling problem with sequence dependent non-anticipatory setup times (SDNAST-JSP), we are given set of n jobs, $J = \{J_1, \dots, J_n\}$, which have to be processed on a set of m machines or resources, $M = \{M_1, \dots, M_m\}$. The processing of a job in a machine is called an operation. Let Ω denote the set of operations, and for an operation $v \in \Omega$, let p_v denote its processing time. If $u \in \Omega$ requires the same machine as v, a setup time s_{uv} is needed to adjust the machine when v is processed right after u. There is also an initial setup time of the machine required by v, denoted s_{0v}, if this is the first operation on that machine. Finding a solution requires to determine a starting time t_v for each operation v subject to a set of constraints: (i) each job has to visit all the machines in a predefined order, (ii) each machine can process only one operation at a time, (iii) setup times are non-anticipatory, meaning that a setup operation can only start after the previous operations both in the job and the machine have finished.

Proceedings of the 19th RCRA workshop on *Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion* (RCRA 2012).
In conjunction with AI*IA 2012, Rome, Italy, June 14–16, 2012.

A solution may be viewed as a feasible processing order σ for all the operations. Given σ, 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 respectively. If $u = PM_v$, the starting time of v can be calculated as $t_v = \max(c_{PJ_v}, c_u) + s_{uv}$ and its completion time as $c_v = t_v + p_v$. The objective is to find a solution σ that minimizes the makespan, i.e. the completion time of the last operation, denoted as $C_{max}(\sigma) = \max_{v \in \Omega} c_v$.

The SDNAST-JSP has recently been considered in [11] where the authors propose a simulated annealing (SA) algorithm and compare it with previous approaches: the hybrid genetic algorithm (HGA) from [12], the variable neighborhood search from [16] and a simple shortest processing time (SPT) rule. The experimental study makes it clear that the proposed SA algorithm outperforms the other methods, thus establishing this method as the new state-of-the-art for the SDNAST-JSP.

In this paper, we propose a new neighborhood structure, termed N_1^E, for this problem. This new structure is incorporated to a hybrid algorithm which combines three metaheuristics: scatter search, tabu search and path relinking. This hybrid algorithm is later evaluated on the same benchmarks used in [11]. The experimental results show that the SS algorithm proposed herein compares favorably with the SA algorithm proposed in [11]. Furthermore, SS establishes new best solutions for all the instances.

The rest of the paper is organized as follows, in Section 2 we describe and formalize the neighborhood structure. Section 3 summarizes the main characteristics of the scatter search, path relinking and tabu search algorithms used. Section 4 reports the results from the experimental study. Finally, in Section 5 we summarize the main conclusions and propose ideas for future work.

2 The neighborhood structure.

Previous to defining the structure N_1^E, we shall propose a solution graph model for the SDNAST-JSP which adequately represents the non-anticipatory property of setup times. It will be a variant of the model defined in [19] for the the same problem with anticipatory setup times. Then, in order to define N_1^E, we will analyze all the single moves that can be done in the solution graph, where a single move is understood as the reversal of the processing order of two consecutive operations in the same machine. The first step is to establish which moves lead to feasible processing orders. Later, we consider the chance that these moves improve the makespan.

2.1 Solution graph model

For the SDNAST-JSP, we propose that a feasible operation processing order σ be represented by an acyclic directed graph G_σ where each node represents an operation of the problem, with the exception of the dummy nodes *start* and *end*,

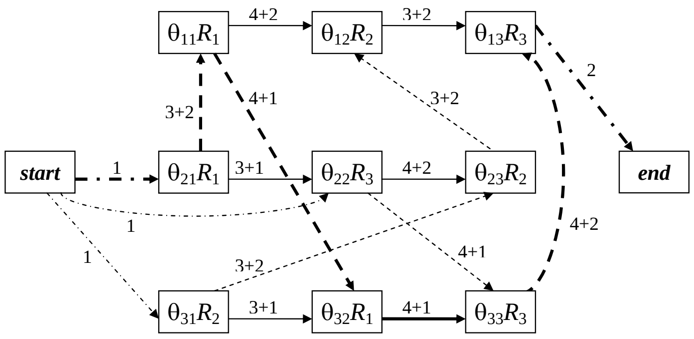


Fig. 1. A feasible schedule to a problem with 3 jobs and 3 machines. Bold-face arcs show a critical path whose length, i.e. the makespan, is 24.

which represent fictitious operations with processing time 0. There are *conjunctive arcs* representing job processing 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 $p_u + s_{vw}$ if $v = PM_w$ (this is the main difference w.r.t. the solution graph defined in [19] for the problem with anticipatory setup times). 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 both in the job and machine processing orders there is an arc (w, end) with weight p_w. Figure 2.1 shows a solution to a problem with 3 jobs and 3 machines.

The makespan of the schedule is the cost of a critical path in G_σ, i.e., a directed path in G_σ from node *start* to node *end* having maximum cost. Nodes and arcs in a critical path are also termed critical. A critical block is maximal subsequence of consecutive operations in a critical path requiring the same machine. Bold-face arcs in Figure 2.1 represent a critical path. Most neighborhood structures proposed for job shop scheduling problems rely on exchanging the processing order of operations in critical blocks with at least two operations [1,18]. The structure proposed in the next subsection includes moves of this type, but as we shall see, in the SDNAST-JSP reversing non-critical arcs may also lead to interesting neighbors.

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:

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

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

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. Clearly, a node v is critical if and only if $C_{max} = r_v + p_v + q_v$.

2.2 Analysis of simple moves

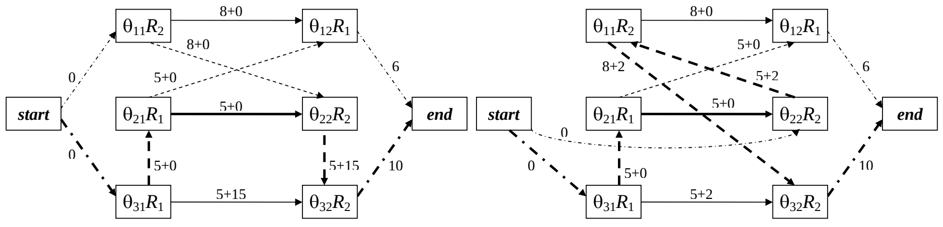
In this paper we focus our attention on single moves involving the reversal of a single disjunctive arc. For the classical job shop scheduling problem, it is well known that reversing a single critical arc always leads to a feasible schedule and that reversing either a critical arc not in the border of a critical block or a non-critical arc does not produce any improvement even if the resulting schedule is feasible. These and other results are the basis for the neighborhood structures designed for that problem [1,18]. The presence of setup times changes the nature of the scheduling problems, so these results are no longer true. For example, reversing a critical arc inside a critical block may produce an improving schedule [19]. Considering non-anticipatory setup times adds further possibilities for improvement, as we shall see in the sequel.

In the presence of setup times, reversing some non-critical arcs can produce immediate improvement. This can be illustrated with an example with 3 jobs and 2 machines. The schedule in Figure 2(a) has a critical path $(\theta_{31}, \theta_{21}, \theta_{22}, \theta_{32})$ with makespan 40. Reversing the non-critical arc $(\theta_{11}, \theta_{22})$ causes the critical arc $(\theta_{22}, \theta_{32})$ to disappear in the neighboring solution in Figure 2(b) where the new makespan is 37.

Having non-anticipatory setup times augments the cases when reversing a non-critical arc improves the makespan. Indeed, if v is a task in a critical path, reversing the disjunctive arcs (PM_{PM_v}, PM_v), (PM_v, v) or (v, SM_v) may produce immediate improvements in the makespan, even if PM_v, SM_v or both are not in that critical path. Figure 3 illustrates the reversal of an arc (PM_v, v) where $v = \theta_{12}$ is the only task in its critical block. Notice that the critical path is the same in both solutions, however the neighbor's makespan is shorter due to the difference in the setup times between operations θ_{12} and θ_{22} and the non-anticipatory nature of the setup times.

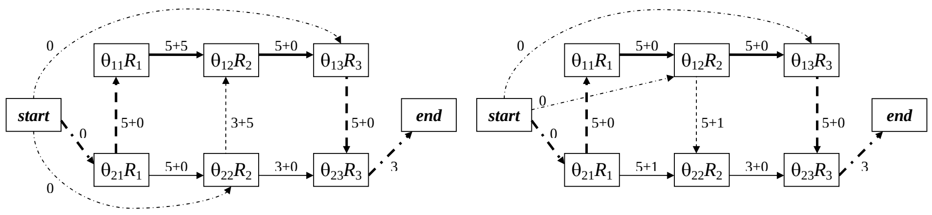
Given the above, we shall consider several single moves for the SDNAST-JSP and makespan minimization. Clearly, this could result in a very large neighborhood but, as we shall see, many of these moves may be discarded based on the feasibility and non-improving conditions given in the next subsections. In principle, the neighborhood of a given solution is built from a critical path selected at random by reversing each of the following single disjunctive arcs (provided that they exist):

1. Critical arcs (v, SM_v) where both v and SM_v are in the same critical block



(a) Initial schedule (b) Neighbor

Fig. 2. A schedule with a makespan of 40, and a neighbor with a makespan of 37 created by reversing a non-critical arc.



(a) Initial schedule (b) Neighbor

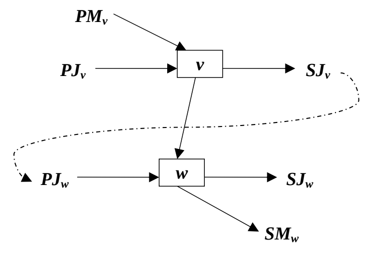
Fig. 3. A schedule with a makespan of 28, and a neighbor with a makespan of 23 created by reversing a non-critical arc.

2. Non-critical arcs (v, SM_v) where one of v and SM_v, but not both, is in a critical path, and arcs (PM_{PM_v}, PM_v) where v is the first task of a critical block.

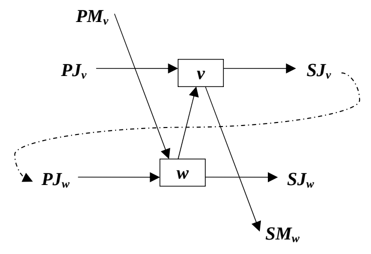
This set of arcs will be denoted by $\mathbf{A}$ and its subsets of critical and non-critical arcs by $C(\mathbf{A})$ and $NC(\mathbf{A})$ respectively. It is easy to prove that reversing an arc which is not in the set $\mathbf{A}$ for some critical path cannot immediately improve the makespan, since any other reversal will not affect the cost of this critical path and, therefore, the makespan in resulting schedule cannot be inferior. In consequence, this structure considers all possible reversals of a single disjunctive arc in the solution graph such that the neighbor may produce an immediate improvement in the makespan. The underlying philosophy here is therefore the same as in the well-known neighborhood structure from [13] for the classical JSP.

2.3 Feasibility conditions

For any a disjunctive arc (v, w) in a solution, a necessary condition for feasibility after reversing (v, w) is that no cycle exists in the resulting solution graph, i.e., that there does not exist an alternative path from v to w in the original



(a) Initial solution



(b) Unfeasible neighbor

Fig. 4. Potential alternative path between two successive operations in a machine sequence v and w that could lead to a cycle after reversing the disjunctive arc (v, w).

solution (see Figure 4(a)). However, a complete search for such paths after a move is clearly expensive. Instead, we propose to use a sufficient condition for feasibility which can be efficiently evaluated, at the cost of discarding some feasible neighbors.

Theorem 1. *Let (v, w) be a disjunctive arc in a solution. An alternative path from v to w does not exist if one of the two following conditions hold:*

1. *(v, w) is a critical arc*
2. *$r_{PJ_w} < r_{SJ_v} + p_{SJ_v} + \min\{s_{xy}/(x, y) \in E, x \in SUC_J(v)\}$*

where E is the set of disjunctive arcs and $SUC_J(v)$ is the set of operations after v in the job sequence.

Proof. If an alternative path from v to w existed, it would have to pass through SJ_v and PJ_w, as indicated in Figure 4, so it would have a larger cost than the arc (v, w). Therefore, if this arc is critical, such a path cannot exist. If (v, w) is not critical but condition (2) is satisfied, it is neither possible that such a path exists, since it would include at least one setup operation from between an operation after v and one operation before w in their corresponding job sequences.

2.4 Non-improvement conditions

Computing the value of the makespan for each neighbor is time consuming. Even when we use estimations for evaluating neighbors, it is interesting to establish easy-to-evaluate conditions for non-improvement that allow to discard uninteresting neighbors beforehand at low cost.

The following result provides a necessary condition for non-improvement of a reversal of a critical arc inside a critical block or a non-critical arc associated to a critical task.

Theorem 2. *Let G_σ be a solution graph, let $(u_1 \ldots u_n)$, $n \geq 2$, be a critical block thereof, and let v be a critical task which does not belong to any critical block with more than two tasks. The solution obtained from G_σ by reversing a critical arc (x, y) does not improve G_σ if the corresponding condition given in Table 1 holds.*

Table 1. Conditions for non-improvement of reversal of arc (x, y). The first column in this table indicates if the arc (x, y) is critical (C) or not (NC).

Arc type	x	y	Condition
C	u_1	u_2	$s_{PM_{u_1}u_1} + s_{u_1u_2} + p_{u_2} + s_{u_2u_3} \leq s_{u_2u_1} + s_{u_1u_3}$ $(n \geq 3)$
C	u_{n-1}	u_n	$s_{u_{n-2}u_{n-1}} + p_{u_{n-1}} + s_{u_{n-1}u_n} \leq s_{u_{n-2}u_n}$ $(n \geq 3)$
C	u_i	u_{i+1}	$s_{u_{i-1}u_i} + s_{u_iu_{i+1}} + s_{u_{i+1}u_{i+2}} \leq$
			$s_{u_{i-1}u_{i+1}} + s_{u_{i+1}u_i} + s_{u_iui+2}$ $(2 \leq i \leq n-2)$
NC	$PM_{PM_{u_1}}$	PM_{u_1}	$s_{PM_{u_1}u_1} \leq s_{PM_{PM_{u_1}}u_1}$ ($PM_{PM_{u_1}}$ not critical)
NC	PM_{u_1}	u_1	$s_{PM_{u_1}u_1} + s_{u_1u_2} \leq$
			$s_{PM_{PM_{u_1}}u_1} + s_{u_1PM_{u_1}} + p_{PM_{u_1}} + s_{PM_{u_1}u_2}$
NC	u_n	SM_{u_n}	$s_{u_{n-1}u_n} \leq s_{u_{n-1}SM_{u_n}} + p_{SM_{u_n}} + s_{SM_{u_n}u_n}$
NC	PM_{PM_v}	PM_v	$s_{PM_vv} \leq s_{PM_{PM_v}v}$ (PM_{PM_v} not critical)
NC	PM_v	v	$s_{PM_vv} \leq s_{PM_{PM_v}v}$
NC	v	SM_v	$s_{PM_vv} \leq s_{SM_vv}$

Proof All conditions are easily derived from a single comparison of longest paths before and after the move through node u_1 in the six first cases and through v in the last three cases.

2.5 Definition of N_1^E

Given the above results, the neighborhood structure N_1^E is defined as follows.

Definition 1 **(N_1^E).** *Let σ be a solution, the neighborhood $N_1^E(\sigma)$ is given by all the solutions obtained from the solution graph G_σ after reversing one of the arcs in the set* **A** *provided that one of the feasibility conditions given in Theorem 1 holds and none of the non-improving conditions given in Theorem 2 hold.*

2.6 Makespan estimation

Computing the 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 SM_w when arc (v, w) is reversed. Hence, we propose an estimation which is based on the *lpath* procedure for the classical JSP from [17]. After reversing arc (v, w) in a schedule σ to obtain σ', if $x = PM_v$ and $y = SM_w$

before the move, the new heads and tails for operations v and w are estimated as follows:

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

Given this, the makespan of σ' can in principle be estimated as the maximum length of the longest paths from node *start* to node *end* through nodes v and w respectively: $Est1(C_{max}(\sigma')) = \max\{r'_w + p_w + q'_w, r'_v + p_v + q'_v\}$. Additionally, when we reverse an arc from $NCA(\mathbf{A})$ of the form (PM_{PM_v}, PM_v) with v the first task of a critical block, the critical path in σ may remain unchanged in σ', even though setup times from PJ_v to v may be different in σ'. In this case a second estimation may be obtained as $Est2(C_{max}(\sigma')) = C_{max}(\sigma) - s_{PM_v v} + s_{PM_{PM_v} v}$ and we take the maximum of the two values $Est1$ and $Est2$ as the final estimated value.

Unlike the procedure *lpath* for the JSP [17] and its extension for the SDST-JSP proposed in [19] and due to the non-anticipatory nature of setup times, the method proposed herein for the SDNAST-JSP does not necessarily return a lower bound of the makespan of the neighboring solution σ'. However, it does yield very good estimations. To assess this, we have conducted an experimental study evaluating 25 million neighbors in different types of instances and we have seen that the estimation coincided with the exact makespan value in 88% of the cases, it was lower in 11% of the cases and only in 1% of the cases was the estimation larger than the actual makespan value.

3 Scatter Search for the SDNAST-JSP

Scatter Search (SS) is an evolutionary population-based metaheuristic first proposed in [2], recognized as good at achieving a proper balance between intensification and diversification in search. It has been successfully applied to a great number of problems, in particular to scheduling [8] [20]. The SS five-method template proposed in [5] has been the main reference for most SS implementations to date, including our proposal, shown in Algorithm 1. It starts by creating an initial set of solutions P which are locally improved using Tabu Search (TS). Then, a reference set $RefSet$ is obtained selecting the "best" solutions from P. The algorithm iterates until a stopping condition is met, this being a given number of iterations without improvement. At each iteration, a pair of solutions from $RefSet$ is combined using Path Relinking (PR) to generate a new solution, which is also improved by TS. Then, the reference set update method is applied. Additionally, if all possible pairs of solutions in $RefSet$ have already been combined, a diversification phase is applied which consists in restarting the search from random solutions together with the best solution reached so far. Further detail on the algorithm is given in the following subsections.

```
Input  A SDNAST-JSP instance
Output  A schedule for the input instance
  (1) Complete the set $P$ up to $PSize$ random trial solutions;
  Apply Tabu Search to every solution of $P$;
  Build the reference set $RefSet$ taking from $P$ good and diverse solutions;
  while not Stop Condition do
    if All pairs of solutions in $RefSet$ were already combined then
      Initiate $P$ with the best solution in $RefSet$ and go to (1);
    Choose two solutions $\sigma_{ini}$ and $\sigma_{end}$ from $RefSet$ not combined yet;
    Combine $\sigma_{ini}$ and $\sigma_{end}$ with Path Relinking to obtain a new solution;
    Apply the improvement method (Tabu Search) to the new solution;
    Update the $RefSet$ with the improved new solution;
  return  The best solution in $RefSet$;
```

Algorithm 1: Scatter Search

3.1 Initial reference set construction

As suggested in [9], the reference set must contain a collection of high quality and diverse solutions. To achieve this, an initial set P is generated with $PSize$ random solutions which are all improved using TS. Then, the reference set $RefSet$ (of size $RSSize$) is built with the $RSSize/2$ best solutions from P and then completed with the remaining $RSSize/2$ most diverse solutions. To add a "diverse solution" we select from P the most distant solution to those solutions already in $RefSet$. The distance between two solutions σ_1 and σ_2 (denoted by $D(\sigma_1, \sigma_2)$) is given by the disjunctive graph distance, or Hamming distance, defined in [10] as the number of pairs of operations requiring the same machine which are processed in different order in σ_1 and σ_2.

3.2 Path Relinking

Path Relinking (PR) is a metaheuristic which combines two solutions, referred to as initial (σ_{ini}) and guiding (σ_{end}) solutions, to obtain a new solution. To do this, starting from the initial solution it repeatedly applies moves so each single move produces a solution which is closer to the guiding solution than the current one. In our algorithm, the initial solution is always the best of the two solutions taken from $RefSet$. In principle, the moves are those of the structure N_1^E, so a single move yields the initial solution one unit closer to or farther from the guiding solution. Since several neighbors of one solution may be at the same distance of the guiding solution, the estimated makespan is used as tie-breaking rule.

In order to escape from local optima, similarly to Tabu Search (TS), a neighbor created by reversing an arc already reversed in the last *tabuPath* iterations is discarded, unless it becomes the closest neighbor so far to the guiding solution. Even with this mechanism, local optima may be so deep that it is very difficult to find a complete path from the initial to the guiding solution using N_1^E. For

```
Input A SDNAST-JSP instance $P$, an initial operation processing order $\sigma_{ini}$ and a
  guiding operation processing order $\sigma_{end}$
Output A solution between $\sigma_{ini}$ and $\sigma_{end}$
  $\sigma \leftarrow \sigma_{ini}$; $dist_{min}, dist_{ini}, dist_{cur} \leftarrow D(\sigma, \sigma_{end})$; $numFails \leftarrow 0$; $TL \leftarrow \emptyset$;
  while $dist_{cur} > dist_{ini}/2$ do
    if $numFails < maxFails$ then
      $\sigma^* \leftarrow \arg\min\{D(\sigma', \sigma_{end}), \sigma' \in N_1^E(\sigma) \wedge (D(\sigma', \sigma_{end}) < dist_{min} \vee$
      $\neg Tabu(\sigma', TL))\}$;
    else
      $\sigma^* \leftarrow N_{PR}(\sigma, \sigma_{end})$;
    Update $TL$ accordingly;
    if $D(\sigma^*, \sigma_{end}) < dist_{min}$ then
      $dist_{min} \leftarrow D(\sigma^*, \sigma_{end})$;
    if $D(\sigma^*, \sigma_{end}) > dist_{cur}$ then
      $numFails \leftarrow numFails + 1$;
    $dist_{cur} \leftarrow D(\sigma^*, \sigma_{end})$; $\sigma \leftarrow \sigma^*$;
  return $\sigma$;
```

Algorithm 2: Path Relinking

this reason, we consider a less restrictive neighborhood structure, N_{PR}, consisting of all single moves leading to a feasible schedule. Thus, if N_1^E cannot reach a solution closer to the guiding solution in $maxFails$ attempts, the neighborhood structure changes to N_{PR} and for the remaining of the search a random neighbor is chosen provided it is closer to the guiding solution than the current solution. The search finishes when the current solution is halfway between the initial and guiding solution. The proposed PR method is detailed in Algorithm 2, where TL denotes the Tabu List used in the algorithm, and $\neg Tabu(\sigma', TL)$ means that the move from σ to σ' is not in TL.

3.3 Tabu Search

Tabu search (TS) is an advanced local search technique first proposed in [3,4] which can escape from local optima by temporarily 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, in particular in scheduling. For example, the $i-TSAB$ algorithm from [14] is one of the best approaches for the classical JSP. Also, in [6] a TS algorithm provides the best results so far for the SDST-JSP with lateness minimization. TS is often used in combination with other metaheuristics as genetic algorithms [7] or scatter search and path relinking [15].

The general TS scheme used in this paper is similar to that proposed in [1]. In the first step the initial solution (provided by the SS) 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 and the cycle checking mechanism also proposed in [1]. TS finishes after a number of *maxImproveIter* iterations without improvement, returning the best solution reached so far.

3.4 Reference set update

A new solution σ is obtained after TS has been applied to the solution returned by the PR algorithm. This solution replaces the worst one in $RefSet$, σ_W, if σ is better than the current best solution or if $C_{max}(\sigma) < C_{max}(\sigma_W)$ and the distance from σ to all solutions in $RefSet$ exceeds a given minimum distance $MinDist$.

4 Experimental study

The purpose of this experimental study is to analyze the proposed SS algorithm and compare it with other methods from the literature. We use the benchmark described in [11], with instances of eight different sizes $n \times m$: 15×15, 20×15, 20×20, 30×15, 30×20, 50×15, 50×20 and 100×20. The processing times are taken from an uniform distribution in $(1, 99)$ and the setup times are taken from uniform distributions in $(1, 25)$, $(1, 50)$, $(1, 100)$ and $(1, 125)$. All combinations of these three factors were considered, generating a total of 10 instances for each combination, so there is a total of 320 instances.

4.1 Parameter tuning and analysis of SS algorithm

For parameter tuning, we have performed a preliminary experimental study across the 320 instances considering different values of the parameters. From this study, we have fixed these parameters as follows: $PSize = 20$, $RSSize = 10$, $tabuPath = 5$, $maxFails = 5$ and $maxImproveIter = 400$. Also, we have set $MinDist = n \times m/10$ experimentally. This configuration has achieved the best average results from all the configurations tested. Figure 5 shows the evolution of the best and average makespan using this configuration and a maximum of 200 iterations of SS without improvements for one of the 30×20 instances. We can see the four times that the diversification phase is activated with a sudden increase in the average makespan of the solutions in $RefSet$.

We have also carried out some experiments to assess the synergy between the metaheuristics. To do this, we compared the results of the SS algorithm (with PR and TS) with those from TS alone. We experimented across 32 instances, one from each combination of the three factors. To achieve similar running times, TS required between 500000 and 2000000 iterations, depending on the instance size. The results show that the average makespan obtained by SS is better than that obtained by TS alone in 31 of the 32 instances. Overall, TS obtained an average makespan about 5% worse than that of SS, the differences being in direct ratio with the size of the instance and the average value of the setup times. From

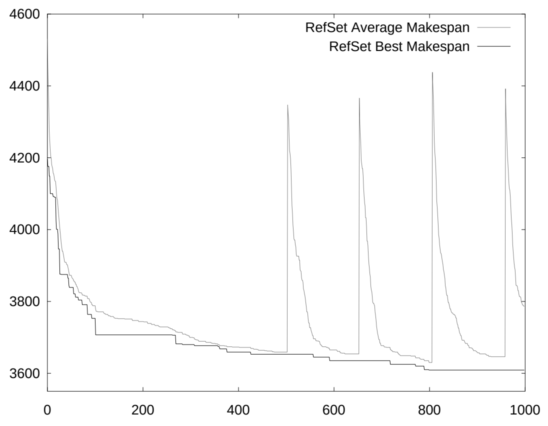


Fig. 5. Evolution of the best and average makespan of $RefSet$ for one run of one 30×20 instance.

these results, we concluded that the combination of the three metaheuristics is better than TS alone. Therefore, we only considered SS in the remaining of the experimental study.

4.2 Comparison with other state-of-the-art methods

We shall compare SS with the simulated annealing method (SA) proposed in [11], which clearly improves the results of the other methods considered in that paper. SA as well as the other methods were implemented in MATLAB 7.0 on a PC with Intel Core 2 Duo at 2.0 GHz and 2 Gb RAM and the best solutions obtained by these methods were reported together with the time taken. For SA, these times were 8.10 seconds for the 15×15 instances, 139.01 seconds for the 100×20 instances, and between 15.58 and 26.68 seconds for the remaining instances, depending on their size. We have implemented SS in C++ on a PC with Intel Core 2 Duo at 2.66 GHz and 2 Gb RAM. Even though the machines are similar, the differences between implementation languages makes a proper comparison of the results difficult. For this reason, we have evaluated SS in two different running conditions. In the first one (SS_S) we have considered short runs of SS by setting the parameters as $PSize = 12$, $RSSize = 6$, $maxImproveIter = 100$, and the stopping condition of SS as 15 iterations without improvement. Then we have done considered longer runs so SS can converge properly (SS_L), with the parameteres indicated in Section 4.1 and the stopping condition of SS set at 200 iterations without improvement. Since SS is an stochastic algorithm, we have run it 30 times on each instance, recording the best and average solutions of the 30 runs. This adds a new difficulty for comparison, as we only know only one solution for each of the other methods, which are also stochastic.

Table 2. Summary of results from SS and SA averaged for groups of instances with the same size

		RPD			Time(s)	
n	m	SA	SS_S	SS_L	SS_S	SS_L
15	15	10.35	4.42	0.90	0.41	8.86
20	15	13.73	6.89	1.67	0.69	14.79
	20	14.01	6.33	1.45	1.09	22.20
30	15	25.16	8.40	1.91	1.57	26.84
	20	28.06	8.52	1.87	2.61	46.10
50	15	23.51	8.55	1.84	4.65	56.60
	20	29.35	9.41	1.91	7.71	100.59
100	20	34.89	9.46	1.62	35.01	274.56
Average		22.38	7.75	1.65		

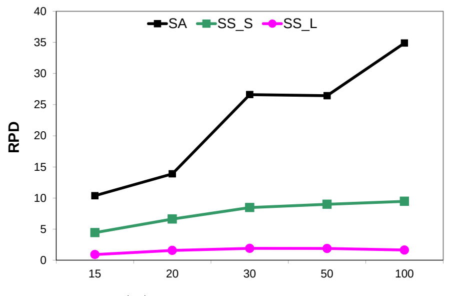
To compare the results of the algorithms, we report the Relative Percentage Deviation (RPD) which is defined as:

$$RPD = \frac{Alg_{sol} - Min_{sol}}{Min_{sol}} \times 100 \qquad (1)$$

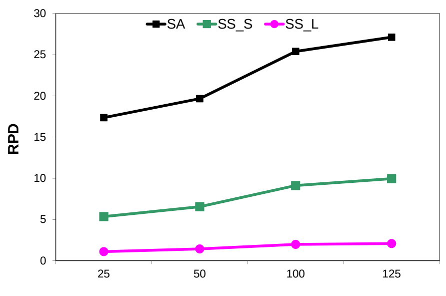
where Alg_{sol} is the value of the objective function obtained by a given algorithm for a given instance, and Min_{sol} is the best-known solution for the instance, including possible new best solutions found with SS. For SA, Alg_{sol} is the solution reported in [11], while for SS it is the average of the 30 solutions.

Table 2 shows the RPD for SA and SS, averaged across each group of instances with the same size, as well as the average runtime in seconds of one run of SS both for short and long runs. We can observe that the RPD obtained by SS is much lower than that obtained by SA, with larger differences as the size of instance increases. Also, the results of SS are much better when it is given enough time to converge, as shown by the differences between the short and long runs. Moreover, SS has achieved better average makespan than the previosly best-known makespan in all of the 320 instances of the benchmark. The detailed results from these experiments, in particular the new best solutions for all the instances, can be downloaded from http://www.di.uniovi.es/tc/Almacen/Test Problems.

For additional comparisons between SS_S and SA, we have done some statistical tests. Since we have different instances, we used a non-parametric test, in particular paired Wilcoxon test, obtaining a p-value of 2.2e-16 which shows that the RPD of SS_S is significantly lower than that of SA. We have also studied the interaction between the solution quality and the different levels of the number of jobs and the different intervals for the setup times. Figures 6(a) and 6(b) show the average RPDs obtained by each algorithm on the different levels of each parameter. We can see that SS greatly improves the other methods for large number of jobs and large upper bounds of the setup times intervals (hence, larger setup times in average).



(a) Number of Jobs



(b) Setup Times

Fig. 6. Interaction between quality of the solutions and instance parameters.

5 Conclusions

We have proposed a scatter search method to minimize the makespan in the job shop scheduling problem with non-anticipatory sequence-dependent setup times. First, we have given a graph model for the solutions. Based on this model, we have defined a neighborhood structure considering all single moves —reversals of a single arc in the solution graph— that may lead to an immediate improvement. The efficiency of this structure relies on feasibility and non-improvement conditions, as well as on an algorithm to estimate the neighbors' makespan, which are also proposed in this paper. In their development we have seen how considering non-anticipatory setup times changes several properties of the problem with respect to the standard SDST-JSP.

The new neighborhood has then been embedded into an algorithm which combines three metaheuristics: tabu search, scatter search and path relinking. This algorithm has been experimentally evaluated on conventional instances, obtaining better solutions than the methods of the current state-of-the-art. In particular, it has established new best solutions for all of the 320 instances in the benchmark.

We believe the main reasons for the good performance of our algorithm are the synergy between the three metaheuristics together with a new neighborhood specifically tailored to the problem with non-anticipatory setup times.

In the future, we plan to extend our approach to other variants or extensions of scheduling problems which are closer to real environments and which usually result harder to solve, for instance, problems with uncertain durations, problems considering alternative objective functions such as total flow time and weighted tardiness, or multiobjective versions of these problems.

Acknowledgements

We would like to thank B. Naderi for providing us with the instances and the best known solution for each one. All authors are supported by Grant MEC-FEDER TIN2010-20976-C02-02.

References

1. M. Dell' Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operational Research*, 41:231–252, 1993.
2. F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.
3. F. Glover. Tabu search–part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.
4. F. Glover. Tabu search–part II. *ORSA Journal on Computing*, 2(1):4–32, 1989.
5. F. Glover. A template for scatter search and path relinking. In J. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution*, volume 1363 of *Lecture Notes in Computer Science*, pages 13–54. Springer, 1998.
6. M. A. González, C. Vela, I. González-Rodríguez, and R. Varela. Lateness minimization with tabu search for job shop scheduling problem with sequence dependent setup times. *Journal of Intelligent Manufacturing*, pages 1 – 14, 2012.
7. M. González, C. R. Vela, and R. Varela. A competent memetic algorithm for complex scheduling. *Natural Computing*, 11:151–160, 2012. 10.1007/s11047-011-9300-y.
8. A. Jain. *A Multi-Level Hybrid Framework for the Deterministic Job-Shop Scheduling Problem.* PhD thesis, Dept. of APEME, University Of Dundee, 1998.
9. M. Laguna and R. Marti. *Scatter search. Methodology and implementation in C.* Kluwer Academic Publishers, 2003.
10. D. C. Mattfeld. *Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling.* Springer-Verlag, 1995.
11. B. Naderi, S. Fatemi Ghomi, and M. Aminnayeri. A high performing metaheuristic for job shop scheduling with sequence-dependent setup times. *Applied Soft Computing*, 10:703–710, 2010.
12. B. Naderi, M. Zandieh, and S. Fatemi Ghomi. Scheduling job shops with sequence dependent setup times. *International Journal of Production Research*, 47:5959–5976, 2009.
13. E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop scheduling problem. *Management Science*, 42:797–813, 1996.
14. E. Nowicki and C. Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8:145–159, 2005.
15. M. G. Resende, C. C. Ribeiro, F. Glover, and R. Martí. Scatter search and path-relinking: Fundamentals, advances, and applications. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 87–107. Springer US, 2010.
16. V. Roshanaei, B. Naderi, F. Jolai, and M. Khalili. A variable neighborhood search for job shop scheduling with set-up times to minimize makespan. *Future Generation Computer Systems*, 25:654–661, 2009.
17. E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.
18. P. Van Laarhoven, E. Aarts, and K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.
19. C. R. Vela, R. Varela, and M. A. González. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics*, 16:139–165, 2010.
20. T. Yamada and R. Nakano. Scheduling by genetic local search with multi-step crossover. In *Proceedings of Fourth International Conference On Parallel Problem Solving from Nature (PPSN IV 1996)*, pages 960–969, 1996.