An advanced scatter search algorithm for solving job shops with sequence dependent and non-anticipatory setups

Miguel A. González [a,*], Camino R. Vela [a]
Ramiro Varela [a] and
Inés González-Rodríguez [b]

[a] *Computing Technologies Group, Department of Computing, Artificial Intelligence Center, University of Oviedo, Spain.*
Campus de Viesques, 33271 Gijón.
E-mail: {mig,crvela,ramiro}@uniovi.es
[b] *Department of Mathematics, Statistics and Computing. University of Cantabria, Spain.*
Los Castros s/n, 39005 Santander.
E-mail: ines.gonzalez@unican.es

In this paper we tackle the makespan minimization in the job shop scheduling problem with sequence-dependent non-anticipatory setup times. To this end, we design a scatter search algorithm which incorporates path relinking and tabu search in its core. The good performance of this algorithm relies on a new neighborhood structure proposed in this paper based on a graph model that incorporates the non-anticipatory characteristic of setup times. To define this structure, we consider all single moves, i.e. reversals of single arcs in the solution graph, and we give some conditions that establish the feasibility and the chance of improvement for the neighbors. We present the results of an experimental study across usual benchmarks to analyze our algorithm and to compare it with the state-of-the-art. In particular, our approach establishes new best solutions for all the instances.
Keywords: job shop scheduling, neighborhood structures, scatter search, path relinking, tabu search, non-anticipatory setup times.

*Corresponding author: Miguel A. González, Computing Technologies Group, Department of Computing, Artificial Intelligence Center, University of Oviedo, Spain. Campus de Viesques, 33271 Gijón. E-mail: mig@uniovi.es

AI Communications
ISSN 0921-7126, IOS Press.

1. Introduction

The Job Shop Scheduling Problem (JSP) has been a research topic over the last decades due to the fact that it is a simple model of many real production environments. However, in some processes the production model has to consider additional characteristics and 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 [1]. These setup operations depend on both the outgoing and incoming jobs, so they cannot be considered as being part of these jobs. An example is presented in [26], where the production process of a paper bag factory is described. It consists of three stages (printing, gluing and sewing) and setup times are required when a machine switches from one bag to another, their values depending on the similarity between bags (color, size, etc.).

Setup times are a relevant characteristic that changes the nature of scheduling problems. This causes that well-known results and techniques for the JSP are not directly applicable to the same problem with setups. Setup considerations in scheduling started in [35], where it was discovered through a simulation study that sequence-dependent setup times play a critical role in the performance of a job shop operating near full capacity. Since then, the problem has received increasing attention by researchers, and a number of approaches to scheduling problems with setup times have been proposed, which have been reviewed in [2].

The setup times can be anticipatory or non-anticipatory. In the first case the setup can begin as soon as the machine is available. In the second case it is required that the precedent operation in the job is completed as well. This consideration changes the nature and some properties of the problem, as we will see through this paper. Non-anticipatory setups are present in several environments. For example, as it is pointed in [29]: "In wood processing, prior to polishing a board of wood, the board itself needs to be attached to the machine and this attachment is in itself a setup where clamps must be added/removed. Therefore, it is a non-anticipatory setup time."

The Job Shop Scheduling Problem with Sequence Dependent and Anticipatory Setup Times (SDST-JSP) was first considered in [5], where a branch and bound algorithm is proposed and evaluated on a well-known benchmark called the BT set. In [3] and [4] the authors propose successful approaches for the same problem. More recently, in [12] and [33] two hybrid approaches are proposed which combine a genetic algorithm with local search procedures, obtaining the best-known solutions for the largest instances of the BT set.

In this paper we confront the Job Shop Scheduling Problem with Sequence Dependent and Non-Anticipatory Setup Times (SDNAST-JSP) with makespan minimization. The SDNAST-JSP has recently been considered in [21] where the authors propose a simulated annealing (SA) algorithm and compare it with previous approaches: the hybrid genetic algorithm (HGA) from [22], the variable neighborhood search from [28] 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.

We propose a new neighborhood structure, termed NSA, for the SDNAST-JSP. This new structure is incorporated to a hybrid algorithm which use a tabu search algorithm as improvement method of a scatter search with path relinking metaheuristic. This hybrid algorithm is then evaluated on the same benchmarks used in [21]. The experimental results show that the SS algorithm proposed herein compares favorably with the SA algorithm proposed in [21]. Furthermore, SS establishes new best solutions for all the instances.

The rest of the paper is organized as follows, in Section 2 we formulate the SDNAST-JSP. In Section 3 we describe and formalize the neighborhood structure. Section 4 summarizes the main characteristics of the metaheuristics used. Section 5 reports the results from the experimental study. Finally, in Section 6 we present the main conclusions and propose ideas for future work.

2. Description of the problem

In the job shop scheduling problem with sequence dependent non-anticipatory setup times (SDNAST-JSP), we are given set of n jobs, $J = \{J_1, \ldots, J_n\}$, which have to be processed on a set of m machines or resources, $M = \{M_1, \ldots, M_m\}$. 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 in the job and the machine have finished.

The setup times we consider do not necessarily verify the triangular inequality. Even though in real scenarios it is expected that the condition $s_{uv} + s_{vw} \geq s_{uw}$ holds for the majority of triplets of operations u, v and w requiring the same machine, there are some situations where this is not the case. Indeed, the instances of the benchmark used in this paper does not verify this condition.

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 right 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 $C_{max}(\sigma) = \max_{v \in \Omega} c_v$.

3. The neighborhood structure

A key component of any local search algorithm is the neighborhood structure used. In the following we define a new neighborhood structure based on reversing a single arc, denoted NSA, for the SDNAST-JSP with makespan minimization, which is in turn based on previous structures for the standard JSP and SDST-JSP.

Previous to defining the structure NSA, 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 [33] for the the same problem with anticipatory setup times. Then, in order to define NSA, 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.

3.1. 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*, 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 [33] 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 in both the job and machine processing orders there is an arc (w, end) with weight p_w. Figure 1 shows a solution graph 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 a maximal subsequence of consecutive operations in a critical path requiring the same machine. Bold-face arcs in Figure 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 [6,32]. 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. The head of an operation v is the cost of the longest path from node *start* to node v, i.e., the starting time of v in the schedule represented by G_σ. The tail q_v is the cost of the longest path from node v to node *end*, minus the processing time of operation in node v. Heads and tails are calculated as follows:

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

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

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 $C_{max} = r_v + p_v + q_v$.

3.2. 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 [6,32]. 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 [33]. Considering non-anticipatory setup times adds further possibilities for improvement, as we shall see in the sequel.

For instance, it is the case that 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 55. 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 50.

Indeed, if v is a task in a critical path, reversing any of the following 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 (u, v) where u and v belong to the same critical block.
2. Non-critical arcs (u, v) where one of u and v, but not both, is in a critical block, or SM_v is the first task in 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 not belonging to 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 of the 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 [23] for the classical JSP. It is important to note that, even if we considered setup times that fulfill the triangular inequality, the set A would be the same.

3.3. Feasibility conditions

For any 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 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. This condition allows us to reduce the computational time by more than 10% while having similar results with respect to using necessary and sufficient conditions based on path search procedures.

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 time from an operation after v in its corresponding job sequence.

□

3.4. 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 $(u_1 \dots u_n)$, $n \geq 2$, 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 $G_{\sigma'}$ obtained from G_σ by reversing the arc (x, y) is not better than G_σ if the corresponding condition given in Table 1 holds.*

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

3.5. Definition of NSA

Given the above results, the neighborhood structure NSA is defined as follows.

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

Notice that due to the proposed sufficient condition for feasibility and the conditions used for discarding non-improving neighbors, we cannot guarantee that the connectivity property holds for the neighborhood NSA. This property holds if an optimal solution can be reached from any point in the search space by using only movements defined in the neighborhood structure. This is a convenient property as, in principle, it reinforces the chance of success in finding an optimal schedule. However, usually has little relevance in the context of metaheuristics, where the number of iterations are limited, and the efficiency of the algorithm usually plays so important a role as the optimality of the final solution. For this reason we have opted to create an structure as efficient as possible, even if the proposed conditions can potentially discard the path to the optimal solution.

3.6. Makespan estimate

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. This process is much more expensive in the SDNAST-JSP than in the classical JSP, as in this last case it is enough to recalculate the head of all operations after v and the tail of all operations before w. Notice that in the SDNAST-JSP we may have to recalculate many additional tails with respect to the JSP (in particular the tails of SM_w, PJ_{SM_w}, and all ancestors of PJ_{SM_w}).

Hence, we propose an estimation method which is based on the *lpath* procedure proposed in [31] for the classical JSP. After reversing (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:

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

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:

$$
\begin{gathered}
Est1(C_{max}(\sigma')) = \\
\max(r'_w + p_w + q'_w, r'_v + p_v + q'_v)
\end{gathered}
$$

Additionally, when we reverse an arc from $NC(\mathbf{A})$ of the form (u, v) with SM_v the first task of a critical block, the critical path in σ may remain unchanged in σ', even though setup times from PJ_{SM_v} to SM_v may be different in σ'. In this case a second estimate may be obtained as

$$Est2(C_{max}(\sigma')) = C_{max}(\sigma) - s_{vSM_v} + s_{uSM_v}$$

Therefore, we can take the maximum of the two values $Est1$ and $Est2$ as the final estimated value.

Unlike the procedure *lpath* for the JSP [31] and its extension for the SDST-JSP proposed in [33] 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'. This is due to the fact that the tails of SJ_v and SJ_w may change from the original schedule to the neighbor.

However, the procedure does yield very good estimates. To assess this, we have conducted an experimental study evaluating 25 million neighbors in different types of instances and we have seen that the estimate 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 estimate larger than the actual makespan value. We also have observed that the accuracy of the estimates was inversely proportional to the problem size and the length of the setup times.

4. Scatter search for the SDNAST-JSP

Scatter Search (SS) is a population-based evolutionary metaheuristic first proposed in [7], which is 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 [17] [25] [36].

The SS five-method template proposed in [10] has been the main reference for most SS implementations to date, including our proposal. This template consists of:

1. A diversification-generation method to generate a collection of diverse trial solutions.
2. An improvement method — in our case, tabu search — to transform a trial solution into an enhanced trial solution.
3. A reference-set update method to build and maintain a reference set consisting of the b "best" solutions found so far (usually the value of b is kept small, e.g. no more than 20). In this case, "best" does not necessarily refer to the value being optimized: solutions are included in the reference set depending not only on their quality but also on their diversity with respect to the other solutions already in the set.
4. A subset-generation method which operates on the reference set to produce several subsets of its solutions, later used to create new combined solutions.
5. A solution-combination method to transform a given subset of solutions produced by the subset-generation method into one or more combined solution vectors. In particular, we shall combine pairs of solutions using path relinking in order to obtain one new solution.

Our proposal is 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 without introducing any new solution to the set, a diversification phase is applied. Further detail on the algorithm is given in the following subsections.

4.1. Initial reference set construction

As suggested in [18], 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 [19] as the number of pairs of operations requiring the same machine which are processed in different order in σ_1 and σ_2.

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 without introducing any new solution to the set **then**
    Initiate P with the best solution in $RefSet$ and go to (1);
  Choose two solutions σ_{ini} and σ_{end} from $RefSet$ not combined yet;
  Combine σ_{ini} and σ_{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

4.2. Subset generation method and diversification phase

For subset generation, we use a simple method that consists in selecting all pairs of solutions in $RefSet$ in a given order. Each pair of solutions is combined to obtain a new solution (as explained in section 4.3), unless they have not changed since the last time they were selected together. $RefSet$ is then updated with the new solution in accordance with the reference set updating method (see section 4.5).

If no new solution is added to $RefSet$ after combining all pairs of solutions, the diversification process is applied: set P is rebuilt starting with the best solution so far, and new $PSize-1$ solutions are generated at random and then transformed by the improvement method. Finally, a new $RefSet$ is obtained from P.

4.3. Solution-combination method

As solution-combination method we use Path Relinking (PR). This procedure 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 NSA, 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 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 NSA. For this reason, we consider a less restrictive neighborhood structure, NSA^*, consisting of all single moves leading to a feasible schedule. Thus, if NSA cannot reach a solution closer to the guiding solution in $maxFails$ attempts, the neighborhood structure changes to NSA^* and for the remaining of the search a random neighbor is chosen provided that it is closer to the guiding solution than the current solution. We have also considered selecting the neighbor from NSA^* with the lowest estimated makespan which is closest to the guiding solution. However we have discarded this last option because in large instances the computational cost was prohibitive, as the number of possible neighbors is very high.

An alternative to the above would be to use only NSA^* in the path relinking algorithm. However, we believe it is better to start using NSA, since it is a more refined structure which guides the path through promising neighbors, and use NSA^* only to complete the path in the case that the other neighborhood gets stuck into local optima.

We have opted to finish the search when the current solution is halfway between the initial and guiding solution, because we have seen experimentally that completing the path is computationally very expensive in large instances, and the results are not better. We have also considered returning the neighbor with the lowest makespan situated between 3/8 and 5/8 of the total distance between the initial and the guiding solution, as it is done in

Input A SDNAST-JSP instance P, an initial operation processing order σ_{ini} and a guiding operation processing order σ_{end}
Output A solution between σ_{ini} and σ_{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**
    $\sigma^* \leftarrow \arg\min\{D(\sigma', \sigma_{end}), \sigma' \in NSA(\sigma) \wedge (D(\sigma', \sigma_{end}) < dist_{min} \vee \neg Tabu(\sigma', TL))\}$;
  else
    $\sigma^* \leftarrow NSA^*(\sigma, \sigma_{end})$;
  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 σ;

Algorithm 2: Path Relinking

[25]. However, the computational cost of this strategy in large instances is very high, as it requires calculating and storing many solutions.

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.

4.4. Improvement method

As improvement method we use tabu search (TS). TS is an advanced local search technique proposed in [8,9] 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, in particular in scheduling. For example, the *i-TSAB* algorithm from [24] is one of the best approaches for the classical JSP. Also, in [13] 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 such as genetic algorithms [14] or scatter search and path relinking [27].

Algorithm 3 shows the tabu search algorithm considered herein, where TL and CL denote, respectively, the Tabu List and the Cycle List. The general scheme is similar to other tabu search algorithms from the literature, like the ones proposed in [6] or [23]. In the first step the initial solution is evaluated. Then, it iterates over a number of steps. In each iteration, the neighborhood of the current solution is built and one of the neighbors is selected for the next iteration. The tabu search finishes after a number of iterations *maxImproveIter* without improvement, returning the best solution reached so far.

The selection rule chooses the neighbor with the lowest estimated makespan, discarding suspect-of-cycle and tabu neighbors. Instead of storing actual solutions in the tabu list, those arcs which have been reversed to generate a neighbor are stored. Thus a new neighbor is marked as tabu if it requires reversing an arc included in the tabu list, unless the aspiration criterion is satisfied, i.e. the estimated makespan is less than that of the current best solution.

The length of the tabu list is usually of critical importance, since it allows for an equilibrium between intensification and diversification in the long term. All TS algorithms try to manage this equilibrium with different proposals based on controlling the number of iterations for which a solution can keep its tabu status. Several studies (see for example [16]) show that a dynamic management usually yields better results than a static one. Indeed, we have conducted some preliminary experiments which confirm this. Here, we use the dynamic length schema and the cycle checking mechanism based on witness arcs used, among others, by Dell' Amico and Trubian in [6].

The run time dedicated to each tabu search must be short so that the overall run time of the scatter search algorithm is not prohibitive. For this reason, we are not using more diversification techniques in the tabu search proposed here.

4.5. Reference set update

A new solution σ is obtained after TS has been applied to the solution returned by PR. This solution replaces the worst one in $RefSet$, σ_W, either 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 $MinDist$, in order to avoid introducing very similar solutions in $RefSet$.

Input A SDNAST-JSP instance P and an initial operation processing order σ^0
Output A solution σ^B with makespan C^B_{max}
$\sigma \leftarrow \sigma^0$; $\sigma^B \leftarrow \sigma$; $C^B_{max} \leftarrow C_{max}(\sigma^B)$;
$improveIter \leftarrow 0$; $TL, CL \leftarrow \emptyset$;
while $improveIter < maxImproveIter$ **do**
$improveIter \leftarrow improveIter + 1$;
$\sigma^* \leftarrow argmin\{C_{max}(\sigma'), \sigma' \in NSA(\sigma) \wedge AspirationC(\sigma') \vee \neg Tabu(\sigma', TL) \wedge \neg Cycle(\sigma', CL))\}$;
Update TL and CL accordingly; $\sigma \leftarrow \sigma^*$;
if $C_{max}(\sigma^*) < C^B_{max}$ **then**
$\sigma^B \leftarrow \sigma^*$; $C^B_{max} \leftarrow C_{max}(\sigma^*)$;
$improveIter \leftarrow 0$;
return σ^B and C^B_{max};

Algorithm 3: Tabu Search

5. Experimental study

The purpose of this experimental study is to analyze the proposed SS algorithm and compare it with other methods from the literature, in particular with the simulated annealing algorithm (SA) proposed in [21], as the current state-of-the-art. We use the benchmark described in that paper, 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 were taken from an uniform distribution in $(1, 99)$ and the setup times s were 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 are a total of 320 instances.

Since SS is a stochastic algorithm, we have run it 30 times on each instance, recording the best, average and worst solutions of the 30 runs. One difficulty for comparison is that for each instance we only know one solution of SA, which is also an stochastic algorithm.

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

$$RPD = \frac{Alg_{sol} - Min_{sol}}{Min_{sol}} \times 100$$

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 [21], while for SS it is the average of the 30 solutions.

5.1. Parameter tuning and analysis of SS algorithm

For parameter tuning, we have performed a preliminary study across 32 instances, one from each combination of the three factors. We considered different values of the parameters and we used similar run times for each configuration tested. Next we are describing each parameter.

$RSSize$ defines the number of solutions of $RefSet$. This number is usually lower than 20, as indicated in [10]. Here we tried $RSSize = 8$ and $RSSize = 10$ because these are typical values in the literature (as an example these values are used in [25] and [36] respectively).

$PSize$ defines the number of solutions of P. In [11] it is suggested that $PSize = max(100, 5 * RSSize)$. However, the diversification phase that we have proposed will need to rebuild the set P several times during an execution. As this process is computationally expensive, we also considered smaller sizes for P. In particular we tested sizes 20, 50 and 100.

$maxFails$ is a parameter of the path relinking algorithm which defines the number of times that we can choose with NSA a neighbor further to the guiding solution, before switching to NSA^*. We considered the values 1 (so it immediately changes to NSA^* when a local optimum is reached), 5 and 20.

$maxImproveIter$ is the stopping condition of the tabu search algorithm, which is a maximum number of iterations without improvement. As TS is embedded in the scatter search core and is executed many times during an execution, we cannot choose high values as in a TS algorithm alone. We considered values 100, 200, 400 and 800.

$MinDist$ is a parameter that controls the minimum distance allowed between solutions in $RefSet$, and hence it should ensure the diversity of the set. It should be obvious that the parameter should depend on the size of the instance, as the average distance between two solutions depends on the instance size. Notice that in an instance with 3 jobs and 3 machines the maximum distance between two solutions is 9, whereas in an instance with 20 jobs and 20 machines it is 3800 (these are calculated by $\frac{n*(n-1)}{2} * m$). As reasonable values, we tested $MinDist = n \times m/5$, $MinDist = n \times m/10$ and $MinDist = n \times m/20$.

From this study, we have fixed these parameters as follows: $RSSize = 10$, $PSize = 20$, $maxFails = 5$, $maxImproveIter = 400$ and $MinDist = n \times m/10$. This configuration has achieved the best average results from all the configurations tested. Table 2 shows a summary of the configurations tested, indicating in bold the selected values.

In particular, Figure 5 details the difference in RPD between different values of the parameter $maxImproveIter$. These values are averaged for instances with the same size. We can see that the value 100 obtains the worst results overall. As for the other values, we can see that the differences in RPD between 200, 400 and 800 are very reduced in the instances of smaller size, but the value 400 was better for the bigger instances.

As stopping criterion for the algorithm we used a maximum of 200 iterations of SS without improvement, since this results in a good convergence pattern. This is shown in Figure 6, which details the evolution of the best and average makespan using this configuration for one of the 30×20 instances. We can also see the four times that the diversification phase is activated with a sudden increase in the average makespan of the solutions in $RefSet$.

5.2. Comparison between SS and TS

It is well known that the tabu search metaheuristic is very efficient in solving scheduling problems (see for example [24], [37] or [13]). Here we have carried out some experiments to assess if the scatter search with path relinking shell is capable of improving the performance of the tabu search alone. To do this, we compared the results of the SS algorithm (with PR and TS) with those from TS alone. To achieve similar running times, TS required between 500000 and 2000000 iterations, depending on the instance size. Figure 7 shows the RPD of each method in each instance. Overall, the average makespan obtained by SS is better than that obtained by TS alone in 31 of the 32 instances. 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 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.

5.3. Comparison with the state-of-the-art

As indicated above, the algorithm SA proposed in [21] has obtained the best results known so far for the 320 instances. The solutions obtained by SA are reported in [21] together with the times taken by the algorithm to reach them: 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. SA was implemented in MATLAB 7.0 and run on a PC with Intel Core 2 Duo at 2.0 GHz and 2 Gb RAM.

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 under two different running conditions.

In the first case (SS_L) we have considered long runs so that SS can converge properly, with the parameters indicated in Section 5.1 and the stopping condition of SS set to 200 iterations without improvement. Then we have done shorter runs (SS_S) by reducing the parameters. In particular we set the values $PSize = 12$, $RSSize = 6$, $maxImproveIter = 100$, and the stopping condition of SS as 15 iterations without improvement. The motivation for the short runs is that, even if the comparison with SA cannot be completely fair, it is likely that if SA is implemented in C++ and run in our machine, the computation time would probably be smaller than the used by SS_L.

Table 3 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.

It is also remarkable that SS has achieved better average makespan than the previously best-known makespan in all of the 320 instances of the benchmark. The detailed results from these experiments and the new best solutions for all the instances can be downloaded from *http://www.di.uniovi.es/tc/*. Table 4 shows the best, average and worst makespan for each method, averaged for each group of instances with the same parameters. For SA we only show one value as we only know the result of one run. It is remarkable that for every group of 10 instances, the average of the worst of the 30 runs of SS_S is better than the average makespan obtained by SA.

For additional comparisons between SS_S and SA, we have done some statistical tests. Since we have different instances, we have 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 8(a) and 8(b) show the average RPDs obtained by each algorithm on the different levels of each parameter. We can see that SS greatly improves SA for large number of jobs and large upper bounds of the setup-times intervals (hence, larger setup times in average).

6. Conclusions

We have proposed a scatter search algorithm 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. While doing this, we have seen how considering non-anticipatory setups changes several properties with respect to the standard SDST-JSP.

The new neighborhood has then been embedded into an hybrid algorithm which combines tabu search with 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 that the main reasons for the good performance of our algorithm are the combination of the diversification provided by the scatter search and path relinking shell combined with the intensification provided by the tabu search, together with the fact that the neighborhood is specifically tailored to the problem with non-anticipatory setup times.

As future work, 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 [15], problems with additional resource types [20], problems considering alternative objective functions such as total flow time [30] or the problem of scheduling discretionary services [34].

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 the Spanish Government under research grants MEC-FEDER TIN2010-20976-C02-02 and MTM2010-16051.

References

[1] A. Allahverdi, C.T. Ng, T.C.E. Cheng and M.Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, **187**, 985–1032, 2008.

[2] V.A. Armentano, M. Felizardo and F. Filho F. Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based GRASP approach. *European Journal of Operational Research*, **183**, 100-114, 2007.

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

[4] E. Balas, N. Simonetti and A. Vazacopoulos. Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling*, **11**, 253–262, 2008.

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

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

[7] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, **8**(1), 156–166, 1977.

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

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

[10] F. Glover. A template for scatter search and path relinking. Editors: J. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers. *Artificial Evolution*, volume 1363 of *Lecture Notes in Computer Science*, 13–54. Springer, 1998.

[11] F. Glover, M. Laguna and R. Marti. *Advances in evolutionary computation: theory and applications*, chap. Scatter search, 519–537. Springer, 2003.

[12] M.A. González, C.R. Vela and R. Varela. A New Hybrid Genetic Algorithm for the Job Shop Scheduling Problem with Setup Times. *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-2008)*, AIII Press, 116–123, 2008.

[13] M.A. González, C.R. 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*, 1–14, 2012.

[14] M.A. González, C.R. Vela and R. Varela. A competent memetic algorithm for complex scheduling. *Natural Computing*, **11**, 151–160. DOI:10.1007/s11047-011-9300-y. 2012.

[15] I. González Rodríguez, C.R. Vela and J. Puente. A Genetic Solution based on Lexicographical Goal Programming for a Multiobjective Job Shop with Uncertainty. *Journal of Intelligent Manufacturing*, **21**, 65–73, 2010.

[16] J.K. Hao, R. Dorne and P. Galinier. Tabu search for frequency assignment in mobile radio networks. *Journal of Heuristics*, **4**(1), 47–62, 1998.

[17] A. Jain. *A Multi-Level Hybrid Framework for the Deterministic Job-Shop Scheduling Problem*. PhD thesis, Dept. of APEME, University Of Dundee, 1998.

[18] M. Laguna and R. Martí. *Scatter search. Methodology and implementation in C.* Kluwer Academic Publishers, 2003.

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

[20] R. Mencía, M. Sierra, C. Mencía and R. Varela. Genetic Algorithm for Job-Shop Scheduling with Operators. *Fourth International Work-Conference on the Interplay between Natural and Artificial Computation, IWINAC*, LNCS-6687, 305–314, 2011.

[21] 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.

[22] 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.

[23] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop scheduling problem. *Management Science*, **42**, 797–813, 1996.

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

[25] E. Nowicki and C. Smutnicki. Some aspects of scatter search in the flow-shop problem. *European Journal of Operational Research*, **169**, 654–666, 2006.

[26] M.L. Pinedo. *Scheduling. Theory, Algorithms, and Systems*, Springer, Third Edition, 2008.

[27] M. Resende, C. Ribeiro, F. Glover and R. Martí. Scatter search and path-relinking: Fundamentals, advances, and applications. *Handbook of Metaheuristics, International Series in Operations Research & Management Science*, **146**, 87–107, 2010.

[28] 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.

[29] R. Ruiz, F. Sivrikaya and T.J. Serifoglu. Modeling realistic hybrid flexible flowshop scheduling problems. *Computers and Operations Research*, **35**(4), 1151–1175, 2008.

[30] M. Sierra and R. Varela. Best-first search and pruning by dominance for the job shop scheduling problem with total flow time. *Journal of Intelligent Manufacturing*, **21**(1), 111–119, 2010.

[31] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, **64**, 278–285, 1993.

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

[33] 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.

[34] X. Wang, N. Policella, S.F. Smith and A. Oddi. Constraint-based methods for scheduling discretionary services. *AI Communications*, **24**, 51–73, 2011.

[35] J.K. Wilbrecht and W.B. Prescott. The influence of setup time on job shop performance. *Management Science*, **16**(4), 391–401, 1969.

[36] T. Yamada and R. Nakano. Scheduling by genetic local search with multi-step crossover. *Proceedings of Fourth International Conference On Parallel Problem Solving from Nature (PPSN IV 1996)*, 960–969, 1996.

[37] C.Y. Zhang, P. Li, Y. Rao and Z. Guan. A very fast TS/SA algorithm for the job shop scheduling problem. *Computers and Operations Research*, **35**, 282–294, 2008.

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	$s_{PM_vv} \leq s_{SM_vv}$

Table 2

Values tested in the parameter tuning. Bold values indicate the best configuration found

Parameter	Values tested
RSSize	8, **10**
PSize	**20**, 50, 100
maxFails	1, **5**, 20
maxImproveIter	100, 200, **400**, 800
MinDist	$n \times m/5$, $\mathbf{n{\times}m/10}$, $n \times m/20$

Table 3
Results from SS and SA averaged for 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		

Table 4
Makespan of SS and SA averaged for instances with the same parameters

			SA		SS_S			SS_L	
n	m	s		Best	Avg	Worst	Best	Avg	Worst
15	15	25	1560	1459	1493	1533	1442	1452	1464
		50	1808	1681	1719	1764	1662	1670	1684
		100	2371	2184	2243	2319	2132	2156	2183
		125	2624	2370	2446	2548	2318	2347	2383
20	15	25	1798	1638	1680	1737	1595	1614	1634
		50	2052	1924	1973	2034	1868	1894	1920
		100	2713	2445	2524	2617	2335	2385	2427
		125	2941	2659	2750	2855	2534	2585	2639
20	20	25	2129	1951	2001	2059	1911	1931	1960
		50	2474	2268	2329	2403	2200	2231	2263
		100	3227	2905	2988	3116	2790	2830	2883
		125	3633	3251	3355	3487	3117	3178	3238
30	15	25	2476	2157	2204	2249	2082	2109	2140
		50	2892	2435	2504	2589	2335	2374	2412
		100	3711	3070	3173	3302	2884	2952	3026
		125	4057	3354	3473	3654	3144	3216	3297
30	20	25	2856	2413	2462	2523	2307	2342	2381
		50	3339	2819	2883	2977	2681	2724	2774
		100	4256	3460	3562	3705	3259	3323	3395
		125	4764	3813	3950	4119	3573	3660	3754
50	15	25	3682	3216	3269	3349	3098	3128	3162
		50	4171	3593	3672	3770	3431	3488	3544
		100	5195	4438	4583	4767	4155	4254	4356
		125	5785	4836	5015	5343	4503	4607	4737
50	20	25	3980	3449	3513	3592	3294	3333	3378
		50	4654	3887	3991	4103	3680	3742	3811
		100	5993	4819	4964	5162	4478	4581	4676
		125	6768	5374	5550	5801	4971	5093	5219
100	20	25	7497	6246	6325	6412	6049	6109	6169
		50	8625	7013	7134	7296	6652	6750	6863
		100	11194	8600	8852	9623	7896	8063	8214
		125	12311	9414	9730	10765	8543	8706	8881
Average			4298	3598	3697	3862	3404	3463	3527

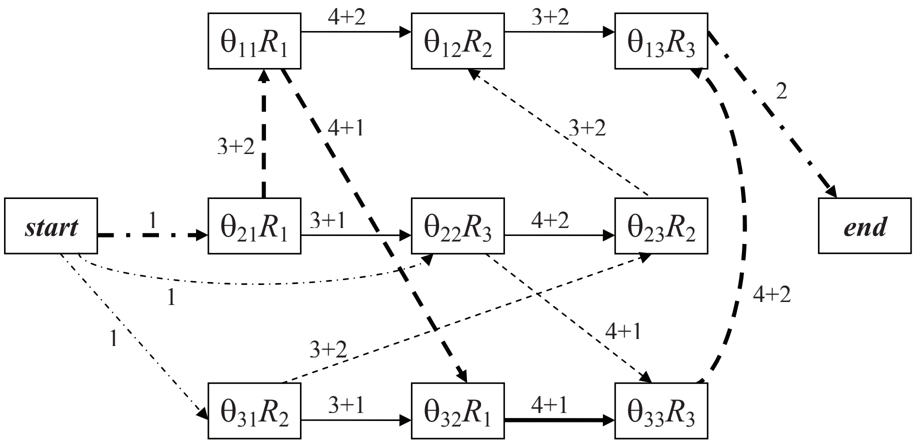
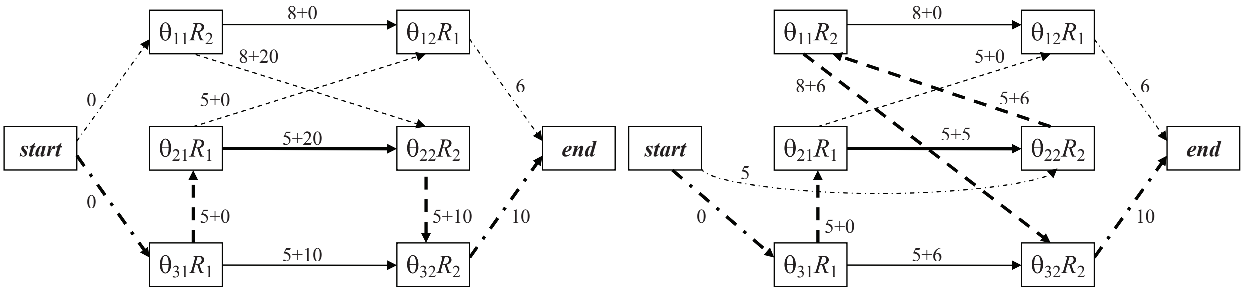
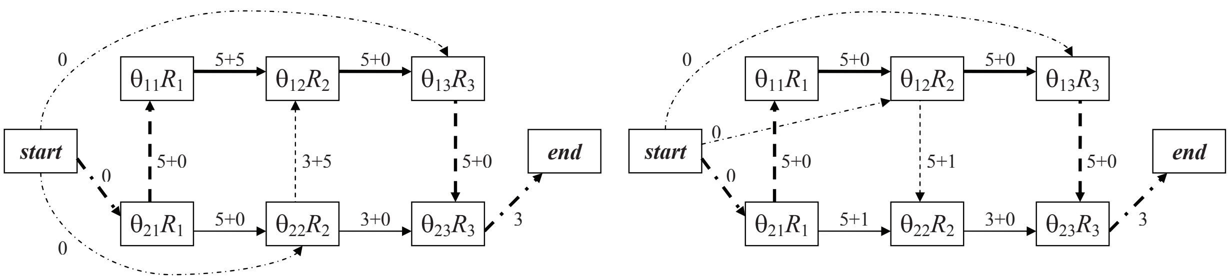


Fig. 1. A feasible schedule to a problem with 3 jobs and 3 machines. Bold-face arcs show a critical path whose length, i.e. the makespan, is 24.



(a) Initial schedule (b) Neighbor

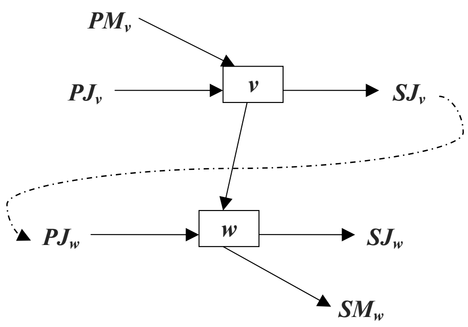
Fig. 2. A schedule with a makespan of 55, and a neighbor with a makespan of 50 created by reversing a non-critical arc.



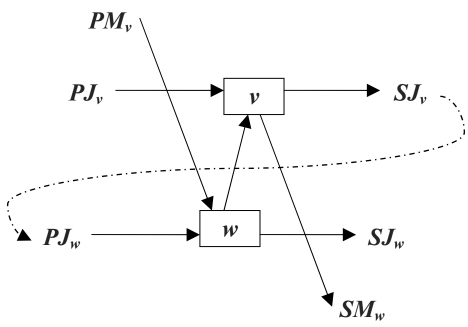
(a) Initial schedule

(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.



(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).

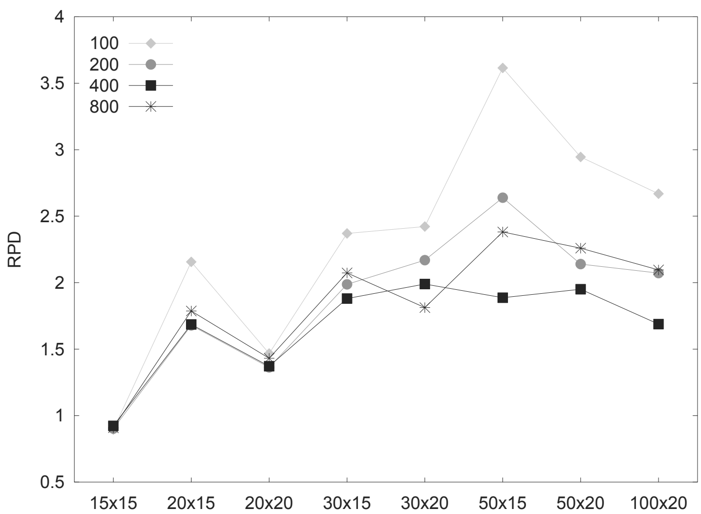


Fig. 5. Solution quality depending on the parameter $maxImproveIter$, averaged for instances with the same size.

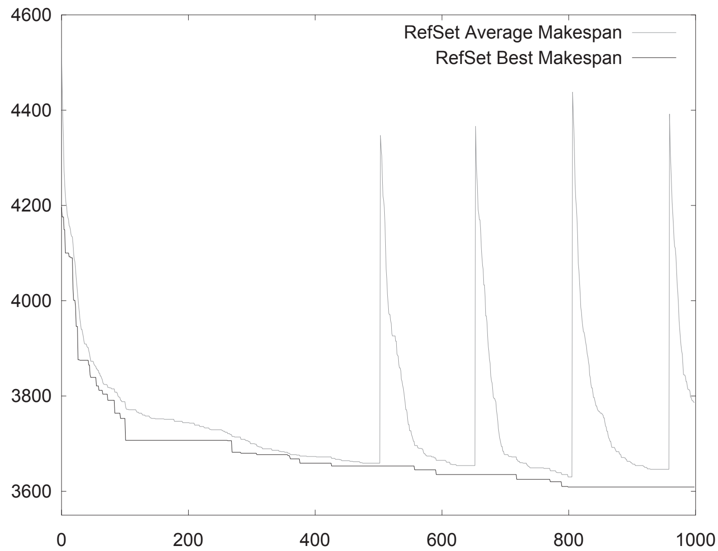


Fig. 6. Evolution of the best and average makespan of $RefSet$ depending on the iteration number, for one run of one 30×20 instance.

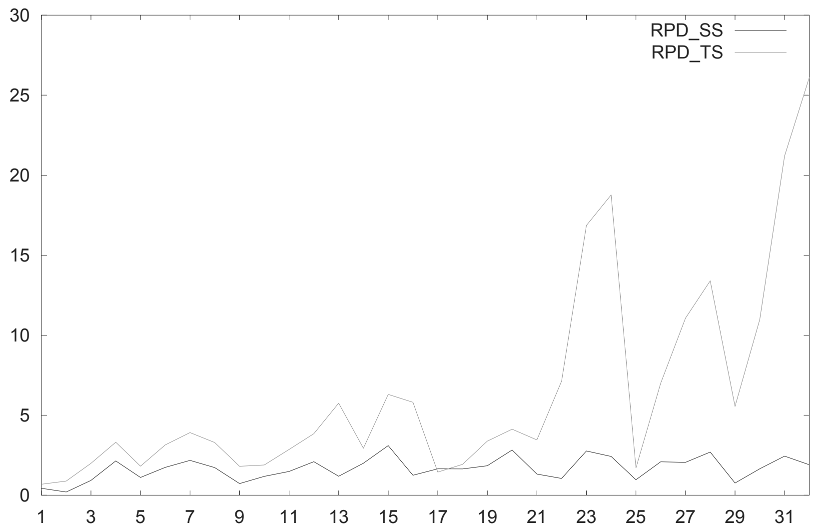
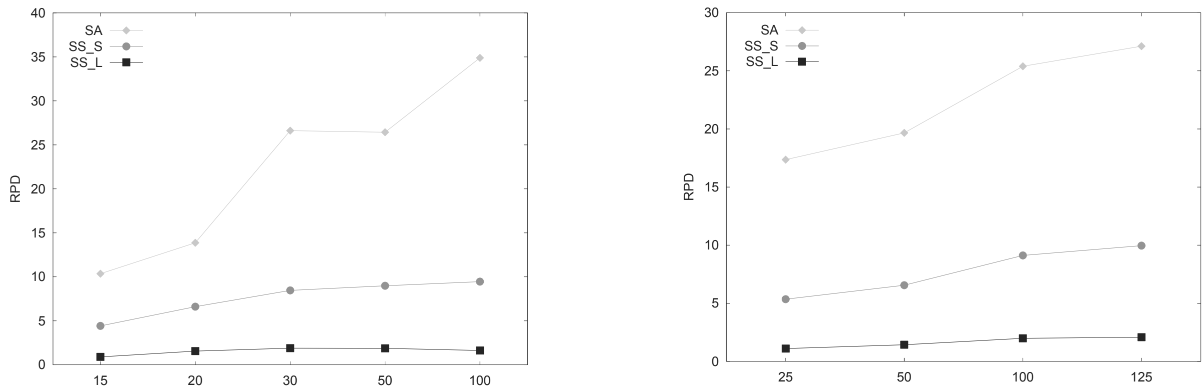


Fig. 7. Comparison of the RPD obtained by SS and TS across 32 instances.



(a) Number of Jobs

(b) Setup Times

Fig. 8. Interaction between quality of the solutions and instance parameters.