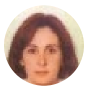
Lateness minimization with Tabu search for job shop scheduling problem with sequence dependent setup times

ARTICLE *in* JOURNAL OF INTELLIGENT MANUFACTURING · AUGUST 2013

Impact Factor: 1.14 · DOI: 10.1007/s10845-011-0622-5

4 AUTHORS:

Miguel Ángel González Fernández
University of Oviedo
17 PUBLICATIONS **95** CITATIONS
SEE PROFILE


Camino Rodríguez Vela
University of Oviedo
62 PUBLICATIONS **287** CITATIONS
SEE PROFILE

Ines Gonzalez Rodriguez
Universidad de Cantabria
41 PUBLICATIONS **146** CITATIONS
SEE PROFILE

Ramiro Varela Arias
University of Oviedo
66 PUBLICATIONS **283** CITATIONS
SEE PROFILE

Journal of Intelligent Manufacturing manuscript No.
(will be inserted by the editor)

Lateness Minimization with Tabu Search for Job Shop Scheduling Problem with Sequence Dependent Setup Times

Miguel A. González · Camino R. Vela · Inés González-Rodríguez · Ramiro Varela

Received: date / Accepted: date

Abstract We tackle the job shop scheduling problem with sequence dependent setup times and maximum lateness minimization by means of a tabu search algorithm. We start by defining a disjunctive model for this problem, which allows us to study some properties of the problem. Using these properties we define a new local search neighborhood structure, which is then incorporated into the proposed tabu search algorithm. To assess the performance of this algorithm, we present the results of an extensive experimental study, including an analysis of the tabu search algorithm under different running conditions and a comparison with the state-of-the-art algorithms. The experiments are performed across two sets of conventional benchmarks with 960 and 17 instances respectively. The results demonstrate that the proposed tabu search algorithm is superior to the state-of-the-art methods both in quality and stability. In particular, our algorithm establishes new best solutions for 817 of the 960 instances of the first set and reaches the best known solutions in 16 of the 17 instances of the second set.

Keywords scheduling · tabu search · setup times · lateness minimization · computational experiments.

A preliminary version of the tabu search method was presented at the 13th Conference of the Spanish Association for Artificial Intelligence, CAEPIA2009, held in Seville, Spain, in 2009 (González et al, 2009)

Camino R. Vela · Miguel A. González · Ramiro Varela
Computing Technologies Group, Department of Computing, Artificial Intelligence Center,
University of Oviedo, Spain. Campus de Viesques, 33271 Gijón.

Miguel A. González
E-mail: mig@uniovi.es

Camino R. Vela
E-mail: crvela@uniovi.es

Ramiro Varela
E-mail: ramiro@uniovi.es

Inés González-Rodríguez
Department of Mathematics, Statistics and Computing. University of Cantabria, Spain.
Los Castros s/n, 39005 Santander. E-mail: ines.gonzalez@unican.es

1 Introduction

In this paper we confront the Job Shop Scheduling Problem with Sequence Dependent Setup Times (SDST-JSP) and maximum lateness minimization. This is a variant of the classical Job Shop Scheduling Problem (JSP) of great interest that has been considered in a number of surveys and research works, (Ovacik and Uzsoy, 1994; Balas et al, 2008; Artigues and Feillet, 2008; Oddi et al, 2009). This kind of problems arise, for instance, in automobile, printing, semiconductor, chemical or pharmaceutical industries (Allahverdi et al, 2008). For instance, (Ovacik and Uzsoy, 1994) presented an extensive study of a digital testing facility of large semiconductor manufacturing, where numerous integrated circuits are tested and the test workcenters have sequence dependent setup times. A benchmark inspired on these problems was proposed therein which has since then been used by a number of researches. Another example is described in (Pinedo, 2008) regarding a paper bag factory producing bags for different uses. The production process 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 considerations are a relevant characteristic that changes the nature of scheduling problems, so well-known results and techniques for the JSP are not directly applicable to the SDST-JSP. Some extensions are however possible, as done for the case of makespan minimization in (Brucker and Thiele, 1996; Vela et al, 2010).

In the sequel, we propose to exploit neighbourhood structures to deal with lateness minimization for the SDST-JSP. We start by defining a new disjunctive graph representation for the SDST-JSP with lateness minimization which is similar to other representations proposed for the classical JSP, for example for lateness minimization (Mehta and Uzsoy, 1998) or considering deadlines and makespan minimization (Balas and Lancia, 1998). Given this new graph, we extend a neighborhood structure proposed for makespan minimization in (González et al, 2008). The resulting structure, denoted N_L^S, is then incorporated to a Tabu Search (TS) algorithm, $TS-N_L^S$. For the experimental study we consider the benchmarks proposed in (Ovacik and Uzsoy, 1994) and in (Brucker and Thiele, 1996), and compare $TS-N_L^S$ with state-of-the-art approaches from (Balas et al, 2008) and (Oddi et al, 2009), illustrating the good performance of our proposal and obtaining new best-known values in some cases.

Setup considerations in scheduling started almost four decades ago in (Wilbrecht and Prescott, 1969), where the authors 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, a number of approaches to scheduling problems have been proposed that take into account setup times. These have been reviewed in (Armentano and Filho, 2007); more recent work on a variety of scheduling problems involving setup times can be found, for example, in (Ying et al, 2011; Fleszar et al, 2011; Huang et al, 2011). The SDST-JSP with makespan minimization was first considered in (Brucker and Thiele, 1996), where the authors proposed a branch and bound algorithm and experimented on the BT set. More recently, in (Vela et al, 2010; González et al, 2008) two hybrid approaches have been proposed that combine a genetic algorithm with local search procedures, obtaining the best-known solutions for the largest instances of the BT set.

In (Uzsoy and Velásquez, 2008) the authors make it clear the relevance of lateness as optimization criteria, motivating the formulation of the shop-floor scheduling problem as that of minimizing maximum lateness (L_{max}), "since this will avoid making some jobs early at the expense of others being extremely delayed". In this paper, the authors establish a relationship between both optimization criteria, L_{max} and C_{max}. In fact, their local search algorithm, termed MEDD, attempts to reduce the makespan at an intermediate step, and this reduction is expected to lead a further reduction in L_{max} as well.

The SDST-JSP with maximum lateness minimization has been considered in a number of research works. For instance, (Ovacik and Uzsoy, 1994) proposed a family of heuristics that use global information to make local decisions at the machine level and experimented with the above mentioned instances, inspired by semiconductor industry. Their results were later improved in (Balas et al, 2008) with the algorithm SB-GLS. More recently, in (Oddi et al, 2009) the authors have proposed an Iterative-Sampling Search (ISS) method, which consists of a constraint-based search embedded within an iterative sampling framework.

Many solving methods for combinatorial optimization use heuristic techniques. One of the most popular heuristics is Tabu Search (TS) (Glover, 1989), with a solid record of good empirical performance in problem solving. In particular, in the field of scheduling, the first TS algorithm for the JSP with makespan minimization was proposed in (Taillard, 1993), outperforming earlier methods based on simulated annealing or shifting bottleneck heuristics. In (Dell' Amico and Trubian, 1993) new neighborhood structures were introduced and used used in a TS algorithm, using new elements such as makespan estimations, a dynamic tabu list which only stores inverted arcs instead of complete solutions or a simple mechanism to avoid cycles. In (Nowicki and Smutnicki, 1996), the authors proposed the TSAB algorithm, introducing the elite solution stack. This algorithm was improved and combined with path-relinking in (Nowicki and Smutnicki, 2005), resulting in the method termed i-TSAB. This has recently been combined with a constructive search procedure in (Beck et al, 2010). In (Grabowski and Wodecki, 2005), some new properties of critical blocks were analyzed and exploited in a TS algorithm. In (Zhang et al, 2008), TS was combined with simulated annealing to introduce new solutions in the elite solution stack; this algorithm and i-TSAB are among the best algorithms for the JSSP with makespan minimization. In (Watson et al, 2006), a thorough study of the i-TSAB algorithm was carried out showing that it is the combination of all the elements of i-TSAB that yields so good results.

TS has also been applied to formal scheduling problems other than JSP with makespan minimization: the parallel machine scheduling problem with setup times and tardiness minimization (Bilge et al, 2004), the JSP with release dates and time lags (DeBontridder, 2005) or the one machine sequencing problem with setup times (Stecco and Cordeau, 2009), to name just a few. Moreover, real scheduling problems have also been solved by means of TS algorithms. For example, in (Yan et al, 2003), the authors propose a solution to an integrated planning and scheduling problem in automobile assembling lines that optimizes several objective functions. In (Meeran and Morshed, 2011), TS is combined with a genetic algorithm and this hybrid approach is able to solve a number of real-life job shop problems.

The rest of the paper is organized as follows. In Section 2 we formulate the SDST-JSP and introduce the notation used across the paper. Then, in Sections 3 and 4, the TS algorithm and the neighborhood structure are described. Section 5 reports results from the experimental study. Finally, in Section 6 we summarize the main conclusions and propose ideas for future work.

2 Description of the problem

The SDST-JSP consists in scheduling a set of jobs $\{J_1, \ldots, J_n\}$ on a set $\{M_1, \ldots, M_m\}$ of machines. Each job J_i consists of a sequence of m operations $(o_{i1}, \ldots, o_{im})$ where operation o_{ij} has a processing time of p_{ij} time units and requires a machine $\mu_{ij} \in \{M_1, \ldots, M_m\}$. A solution to the problem is a schedule, i.e. a starting time for each one of the operations, that is subject to a set of constraints. There are *predecence constraints*, so the operations of each job must be sequentially scheduled and *capacity constraints*, whereby each operation requires the uninterrupted and exclusive use of the machine for its whole processing time. A time may be needed to adjust a machine between two consecutive operations o_{ij} and o_{kl}, which is called a setup time and denoted s_{ijkl}. Here, we consider sequence-dependent setup times, meaning that the machine's setup time for a particular operation is determined by not only by that operation but also by the previous operation that the machine is currently setup for. Job J_i may also have a due date, denoted d_i, a deadline before which all operations of J_i should be completed. The objective is to find a schedule, which, besides being *feasible* (i.e. all constraints hold), is *optimal* according to some criterion, in this paper that the *maximum lateness* is minimal.

Let p, μ and s denote matrices representing processing times, machine requirements and setup times respectively, and let σ be a feasible operation processing order, i.e., a lineal ordering of operations which is compatible with a processing order of operations that may be carried out so that all constraints hold. A feasible schedule may be derived from σ using a *semi-active schedule builder* as follows. Let $C_i(\sigma, p, \nu, s)$ denote the completion time of job J_i and let $T_{ij}(\sigma, p, \mu, s)$ and $C_{ij}(\sigma, p, \mu, s)$ denote respectively the starting and completion times of operation o_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, m$. These times may be obtained as follows:

$$C_i(\sigma, p, \mu, s) = C_{im}(\sigma, p, \mu, s) \tag{1}$$

$$T_{ij}(\sigma, p, \mu, s) = \max\{C_{i,(j-1)}(\sigma, p, \mu, s), C_{rs}(\sigma, p, \mu, s) + s_{rsij}\} \tag{2}$$

$$C_{ij}(\sigma, p, \mu, s) = T_{ij}(\sigma, p, \mu, s) + p_{ij} \tag{3}$$

where rs in equation (2) denotes the indices of the operation preceding o_{ij} in its machine according to the processing order σ. $C_{i0}(\sigma, p, \mu, s)$ is assumed to be zero and, analogously, $C_{rs}(\sigma, p, \mu, s)$ is taken to be zero if o_{ij} is the first operation to be processed in the corresponding machine. The *maximum lateness* is the maximum delay of jobs beyond their due dates:

$$L_{max}(\sigma, p, \mu, s) = \max_{1 \leq i \leq n} \{C_i(\sigma, p, \mu, s) - d_i\} \tag{4}$$

For the sake of a simpler notation we may write, for instance, $L_{max}(\sigma)$ when the problem (hence p, μ, and s) is fixed or even L_{max} when no confusion is possible regarding the operation order.

This problem is denoted by $J|s_{ij}|L_{max}$ according to the $\alpha|\beta|\gamma$ notation proposed in (Graham et al, 1979).

2.1 The disjunctive graph model representation

The disjunctive graph is a common representation in scheduling, its exact definition depending on the particular problem. For the $J|s_{ij}|L_{max}$ problem, we propose that it be represented by a directed graph $G = (V, A \cup E \cup I_1 \cup I_2 \cup I_3)$. Each node in set V represents an operation of the problem, with the exception of the dummy nodes 'start', end_i $1 \leq i \leq n$ and 'end', which represent fictitious operations. Arcs in A are called *conjunctive arcs* and represent precedence constraints while arcs in E are called *disjunctive arcs* and represent capacity constraints. Set E is partitioned into subsets E_i, with $E = \cup_{j=1,\ldots,m} E_j$, where E_j corresponds to machine M_j and includes two directed arcs (v, w) and (w, v) for each pair v, w of operations requiring that machine. Each arc (v, w) in A is weighted with the processing time of the operation at the source node, represented by p_v, and each arc (v, w) of E is weighted with

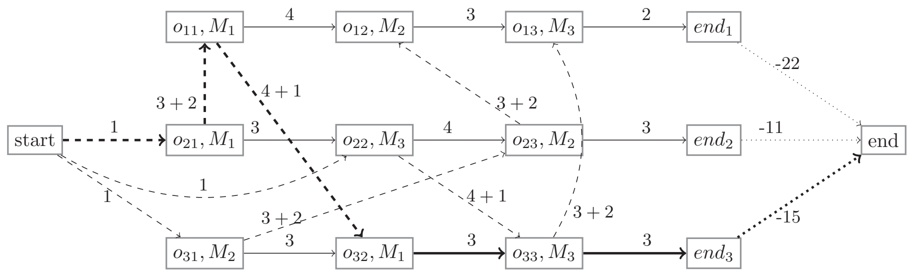


Fig. 1 A feasible schedule to a problem with 3 jobs and 3 machines. Bold face arcs show a critical path whose length, i.e. the maximum lateness, is 2.

$p_v + s_{vw}$ where s_{vw} denotes the setup time between the operations represented by nodes v and w. Set I_1 includes arcs of the form (start, v) for each operation v of the problem, weighted with s_{0v}. Set I_2 includes arcs (u, end_i), being u the node representing operation o_{im}, $1 \leq i \leq n$, weighted with p_u. Finally, arcs in I_3 are of the form (end_i,end) and are weighted with $-d_i$. This model for the problem $J|s_{ij}|L_{max}$ is a natural extension of the disjunctive models proposed in (Mehta and Uzsoy, 1998) and (Vela et al, 2010) for the problems $J||L_{max}$ and $J|s_{ij}|C_{max}$ respectively.

A feasible operation processing order is represented by an acyclic subgraph G_s of G, $G_s = (V, A \cup H \cup L_1 \cup I_2 \cup I_3)$, where $H = \cup_{j=1\ldots m} H_j$, H_j being a minimal subset of arcs of E_j defining a processing order for all operations requiring M_j and where L_1 consists of arcs (start, v_j), $j = 1 \ldots m$, v_j being the first operation of H_j. Finding a solution can thus be reduced to discovering compatible orderings H_j, or partial schedules, that translate into a solution graph G_s without cycles. Figure 1 shows a solution to a problem with 3 jobs and 3 machines; continuous arcs belong to A or I_2, discontinuous arcs belong to H or L_1 and dotted arcs correspond to I_3.

The maximum lateness of the schedule is the cost of a critical path in G_s, i.e., a directed path in G_s from node ‘start’ to node ‘end’ having maximum cost. Nodes and arcs in a critical path are also termed critical. A critical path may be represented as a sequence of the form ‘start’, $b_1, \ldots, b_r$, end_i, ‘end’, $1 \leq i \leq n$, where each b_k, $1 \leq k \leq r$, is a critical block, a maximal subsequence of consecutive operations in the critical path requiring the same machine. The concepts of critical path and critical block are of major importance for job scheduling problems due to the fact that most formal properties and solution methods rely on them. Indeed, according to the so-called so-called properties of elimination for classical JSP, in order to obtain a processing order with lower makespan at least one of the operations in a critical block of the current schedule has to be moved before (after) the first (last) operation of the block (Grabowski and Wodecki, 2005). These properties have given rise to a number of neighborhood structures based on exchanging the order of either adjacent operations (Matsuo et al, 1988; Taillard, 1993; Van Laarhoven et al, 1992; Nowicki and Smutnicki, 1996) or non adjacent operations (Dell’ Amico and Trubian, 1993; Balas et al, 2008) requiring the same machine.

In order to simplify expressions, we define the following notation for a feasible schedule. Given a solution graph G_s, the head of an operation v, denoted r_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_s. 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. Let PJ_v and SJ_v denote respectively the predecessor and successor of v in the job sequence, and PM_v and SM_v the predecessor and successor of v in its machine sequence. We take node 'start' to be PJ_v for the first operation of every job and PM_v for the first operation to be executed in each machine; note that $p_{start} = 0$. Then, the head of every operation may be computed as follows:

$$\begin{aligned} r_{start} &= 0 \\ r_v &= \max(r_{PJ_v} + p_{PJ_v}, r_{PM_v} + p_{PM_v} + s_{PM_v v}) \\ r_{end_i} &= r_v + p_v,\ (v, end_i) \in I_2, 1 \leq i \leq n \\ r_{end} &= \max_{1 \leq i \leq n} \{r_{end_i} - d_i\} \end{aligned} \tag{5}$$

Similarly, for $1 \leq i \leq n$, we take node 'end' as SJ_{end_i} for every job i, node end_i as SJ_v for the last task of job i, and $p_{end_i} = -d_i$ (note that $p_{end} = 0$). Then, the tail of every operation is computed as follows:

$$\begin{aligned} q_{end} &= q_{end_i} = 0 \\ q_v &= \begin{cases} \max(q_{SJ_v} + p_{SJ_v}, q_{SM_v} + p_{SM_v} + s_{vSM_v}) & \text{if } SM_v \text{ exists} \\ q_{SJ_v} + p_{SJ_v} & \text{otherwise} \end{cases} \\ q_{start} &= \max_{v \in SM_{start}} \{q_v + p_v + S_{0v}\} \end{aligned} \tag{6}$$

A node v is critical if and only if $L_{max} = r_v + p_v + q_v$.

With the proposed graph representation the maximum lateness minimization may be transformed into a makespan minimization one. Indeed, if we give the arcs $(end_i$,end$)$ null weight instead of $-d_i$, this results into a disjunctive graph equivalent to the one defined for makespan minimization. Also, we shall see that neighborhood structures relying on moves in a critical path initially defined for makespan may be naturally extended to the maximum lateness problem.

3 Tabu Search for the SDST-JSP

Tabu search (TS) is an advanced local search technique 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. With a solid record of good empirical performance in problem solving, to our knowledge it has never been used to tackle the SDST-JSP with lateness minimization.

Algorithm 1 shows the tabu search algorithm considered herein, where TL, CL and ESL denote, respectively, the Tabu List, the Cycle List and the Elite Solution Stack. The general scheme is similar to other tabu search algorithms from the literature (Nowicki and Smutnicki, 1996), (Dell' Amico and Trubian, 1993). In the first step the initial solution is generated and 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. After a number of iterations without improvement, the search is restarted from a previous solution taken from the elite solution stack, *ESL*. The tabu search finishes either after a number of iterations *maxGlobalIter*, or when the elite solution stack becomes empty, returning the best solution reached so far. In the remaining of this section we describe the algorithm in more detail, while the neighborhood structure will be described in depth in the next section.

```
Input  A SDST-JSP instance $P$
Output  An operation processing order $\sigma$ for $P$
  $\sigma^0 \leftarrow Initialsolution$; $\sigma \leftarrow \sigma^0$; $\sigma^B \leftarrow \sigma$; $L^B_{max} \leftarrow L_{max}(\sigma^B)$;
  $globalIter, improveIter \leftarrow 0$; $TL, CL \leftarrow \emptyset$; $ESL \leftarrow \sigma$;
  while $globalIter < maxGlobalIter$ do
      if $improveIter = maxImproveIter$ then
          if $ESL = \emptyset$ then
              return $\sigma^B$ and $L^B_{max}$,
          else
              $\sigma \leftarrow pop(ESL)$; $improveIter \leftarrow 0$; $TL, CL \leftarrow \emptyset$;
      $globalIter \leftarrow globalIter + 1$; $improveIter \leftarrow improveIter + 1$;
      $\sigma^* \leftarrow argmin\{L_{max}(\sigma'), \sigma' \in N^S_L(\sigma) \wedge AspirationC(\sigma') \vee$
          $\neg Tabu(\sigma', TL) \wedge \neg Cycle(\sigma', CL))\}$;
      Update $TL$ and $CL$ accordingly; $\sigma \leftarrow \sigma^*$;
      if $L_{max}(\sigma^*) < L^B_{max}$ then
          $\sigma^B \leftarrow \sigma^*$; $L^B_{max} \leftarrow L_{max}(\sigma^*)$; $improveIter \leftarrow 0$; $push(\sigma^*, ESL)$;
  return $\sigma^B$ and $L^B_{max}$;
```

Algorithm 1: The Tabu Search Algorithm

We propose to start the search from a randomized semiactive schedule, generated using the Serial Schedule Generation Scheme (SerialSGS) from (Artigues et al, 2005). This schedule builder chooses one of the available operations at random and assigns to it the earliest starting time satisfying all constraints with respect to the previous scheduled operations. An operation is available when its predecessor in the job sequence and its predecessor in the machine sequence has already been scheduled. The process continues until all operations are scheduled. SerialSGS produces active schedules, provided that the triangular inequality for the setup times holds, i.e. $s_{uv} + s_{vw} \geq s_{uw}$ for all operations u, v and w requiring the same machine, otherwise it can produce even non semi-active schedules. Since one of the benchmarks used in our experimental study does not satisfy this property, we have opted to use the following repair algorithm after SerialSGS in order to transform the generated schedule into a semiactive one: the operations are visited in ascending processing time order and every one is rescheduled at the earliest processing time possible without delaying any other operation.

The above way of generating the initial solution is simpler than that used by TSAB (Nowicki and Smutnicki, 1996), but more sophisticated than the priority rule used in (Serifoglu and Ulusoy, 1999). It presents the advantage of quickly obtaining solutions with acceptable quality and variability, a fact illustrated by the experimental results in Section 5.

The selection rule chooses the neighbor with the lowest estimated maximum lateness, discarding suspect-of-cycle and tabu neighbors. Instead of storing actual solutions in the tabu list, it stores those arcs which have been reversed to generate a neighbor. A new neighbor is thus marked as tabu if it requires reversing at least one arc from the tabu list, unless the aspiration criterion is satisfied, i.e. if the estimated maximum lateness 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 short 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, for instance in (Hao et al, 1998), show that a dynamic handling usually yields better results than a static one. Indeed, we have conducted some preliminary experiments which confirm this. Some authors, such as (Nowicki and Smutnicki, 1996), (Bilge et al, 2004) or (Serifoglu and Ulusoy, 1999), propose to vary the number of iterations for which a move is tabu depending on the number of iterations for which the solution has not improved. Other authors, for example (Dell' Amico and Trubian, 1993), propose to use a tabu list of dynamic length for this purpose, an schema we have adopted herein. Additionally, we use a a cycle checking mechanism based on witness arcs similar to that proposed in that same paper.

In order to manage long term intensification, we propose to use long term memory which allows to recover elite solutions. This is a standard strategy, proposed in (Nowicki and Smutnicki, 1996) for the TSAB algorithm and widely used with some changes since then, for instance in (Bilge et al, 2004) or (Serifoglu and Ulusoy, 1999). In our case, after *maxImproveIter* iterations without improvement, the current best solution is replaced with one extracted from L. We set $maxImproveIter = maxGlobalIter/Size(ESL)$ which is large enough for the elite solution stack not to get empty before *maxGlobalIter* iterations.

4 The neighborhood structure.

A key component of any tabu search algorithm is the neighborhood structure used therein. In the following we define for the SDST-JSP with lateness minimization a new neighborhood structure, denoted N_L^S. It adapts to this problem the structure termed N^S in (González et al, 2008), which was defined for the SDST-JSP with makespan minimization and was in turn based on previous structures for the standard JSP.

It is well-known that the concepts of critical arc and critical block play a central role in the definition of neighborhood structures for the JSP. For example, the neighborhood defined in (Van Laarhoven et al, 1992) is motivated by the following two results: (1) reversing a critical arc in a solution graph G cannot lead to a cycle, and (2) if the reversal of a non critical arc in G leads to an acyclic graph G', a longest path in G' cannot be shorter than a longest path in G. Also, the local search proposed in (Nowicki and Smutnicki, 1996) is based on a neighborhood defined in (Matsuo et al, 1988) that avoids non-improving neighbors based on other results given in (Van Laarhoven et al, 1992), namely (3) reversing an arc inside a critical block cannot lead to an improving solution. On the other hand, in (Grabowski and Wodecki, 2005), it is suggested that this last neighborhood is so small that tabu search requires too many iterations to converge. It is easy to prove that the result (2) above holds for the the SDST-JSP if the setup times fulfill the triangular inequality, but if the triangular inequality does not hold it is possible to obtain an improving schedule by reversing a non critical arc. However, considering all non critical arcs as candidates for reversal would be extremely time consuming. At the same time, we assume that 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. For these reasons, in the proposed neighborhood structure we will restrict the move choices to critical arcs only. Finally, result (1) above does not hold as a trivial consequence of Theorem 1 given below and, from the Theorem 2, the result (3) above is not true for SDST-JSP.

Given these considerations, we have defined a neighborhood for the SDST-JSP with lateness minimization that in principle has to consider a very large number of moves, but where a great number of them can be discarded due to non-feasibility or non-improving conditions. Given a solution, its neighborhood is built by considering all critical blocks within each critical path and attempting to alter the relative order of operations within every block. In principle, for a block b, every operation in this block should be moved to any other position in b, including the begin and the end of the block. However some of these possible moves may lead to non-improving or even unfeasible neighbors.

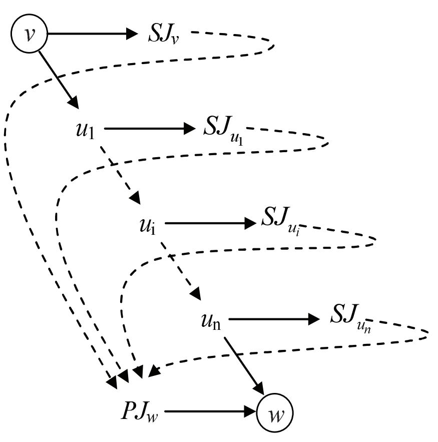


Fig. 2 Potential alternative paths between two operations v and w in a critical block that could lead to a cycle after moving operation w before operation v.

Suppose that, for the initial solution, there are two operations v and w in a critical block, v preceding w, and there also exists an alternative path in the solution graph from v to w through PJ_w. In this case, if the move shifts w before v, a cycle will appear in the graph for the resulting neighbor, making it unfeasible. The following result provides a sufficient condition for feasibility after a move.

Theorem 1 *Given a critical block of the form* $(b_1\ v\ b_2\ w\ b_3)$*, where* b_i *are sequences of operations and* B_i *the set of operations of* b_i*,* $1 \le i \le 3$*, if the following condition is satisfied*

$$\forall u \in \{v\} \cup B_2, \quad r_{PJ_w} < r_{SJ_u} + p_{SJ_u} + \min\{s_{xy}/(x,y) \in E, x \in SUC_J(u)\} \tag{7}$$

then an alternative path from v *to* w *does not exist. Here,* $SUC_J(u)$ *denotes the set of operations after* u *in the job sequence.*

Proof Let us denote $b_2 = (u_1 \dots u_i \dots u_n)$. The only alternative paths from v to w are those indicated in Figure 2 and these paths would exist after moving w to v as well. Since each potential alternative path includes at least a setup time, none of these paths can exist if condition (7) holds.

Notice that a necessary condition for feasibility is that no cycles exist in the resulting solution graph, i.e., that none of the alternative paths shown in Figure 2 exist. However, a thorough check of the existence of such paths may turn to be computationally very expensive. It is therefore preferable to establish and test a sufficient condition for feasibility which can be easily evaluated, despite the possibility of discarding some feasible neighbours as well. In particular, the condition provided by equation (7) can be evaluated in constant time for each pair of operations v and w. Moreover, in just one iteration from w to the begin (end) of the critical block, all feasible positions to insert w can be calculated. This iteration finishes when an unfeasible position is reached. Hence, all candidate neighbours are obtained in polynomial time.

Additionally, computing the lateness value for each neighbor is time consuming. It is therefore 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.

Theorem 2 *Let G_s be a solution graph and $(b_1\ v\ b_2\ w\ b_3)$ a critical block thereof, where b_i, $1 \leq i \leq 3$, are sequences of operations of the form $b_i = (u_{i1} \dots u_{in_i})$ with $n_1, n_3 \neq 0$. Assuming the solution G'_s obtained from G_s by moving w just before v is feasible, G'_s does not improve G_s if the following condition holds*

$$s_{u_{1n_1}v} + s_{u_{2n_2}w} + s_{wu_{31}} \leq s_{u_{1n_1}w} + s_{wv} + s_{u_{2n_2}u_{31}}. \tag{8}$$

If $n_2 = 0$, u_{2n_2} should be substituted by v in (8).

Proof Condition (8) derives easily from a single comparison of longest paths through node u_{31} before and after the move. Notice that after the move the paths from node 'start' to b_1 and from b_3 to node 'end' do not change. The only changes take place in the path from b_1 to b_3 and they are due to the differences in the added and removed setup times.

Given the above results, the neighborhood structure N_L^S is defined as follows.

Definition 1 (N_L^S) For a given solution, let b be a critical block of any critical path and let v be an operation within block b. In a neighboring solution, v is moved to another position in b, provided that condition (8) does not hold and that the sufficient condition of feasibility (7) is preserved.

4.1 Maximum lateness estimation.

For the sake of efficiency, the selection rule is based on estimating rather than actually computing the maximum lateness of all neighbors. For this purpose, we adapt the procedure *lpathS* given in (Vela et al, 2010) for makespan estimation, which in turn extends the *lpath* rule given for the JSP in (Taillard, 1993). This procedure takes as input a sequence $(Q_1 \dots Q_q)$ of operations obtained after a move, all of them requiring the same machine, and where $(Q_1 \dots Q_q)$ is a permutation of the sequence $(O_1 \dots O_q)$ before the move. For each $i = 1, \dots, q$, *lpathS* estimates the cost of the longest path from node 'start' to node 'end' through Q_i. The maximum of these values is taken as the estimate of the maximum lateness for the neighboring schedule. In particular, for N_L^S, if w is moved before v in a block of the form $(b_1\ v\ b_2\ w\ b_3)$, the input sequence for *lpathS* should be $(w\ v\ b_2)$.

The procedure *lpathS* produces very good estimates of the maximum lateness after a move. To assess this feature we have evaluated 3000000 neighbors, observing that the estimate coincided with the exact value in 96% of the cases, it was smaller in 3% of the cases and it was larger only in 1% of the cases.

5 Experimental Study

The purpose of this section is to assess the proposed $TS - N_L^S$ algorithm and compare it with state-of-the-art algorithms from the literature, namely SB-GLS (Balas et al, 2008) and ISS (Oddi et al, 2009).

First, we shall adopt the instances proposed in (Ovacik and Uzsoy, 1994), well-known for being hard to solve. They are categorized in six different benchmark sets: *i*305, *i*315, *i*325, *i*605, *i*615, and *i*625, according to the parameters used to generate due date values: τ, called *tardiness*, and R, called *spread*. The first represents the percentage of jobs expected to be tardy with values equal to 0.3 and 0.6, corresponding to loose and tight due dates respectively. The latter models different due date variation levels, with values equal to 0.5, 1.5 or 2.5. Each benchmark contains 160 problem instances, divided in 8 subclasses of 20 instances each, according to the number of machines ($m \in \{5, 10, 15, 20\}$) and jobs ($n \in \{10, 20\}$) involved. In total there are 960 instances.

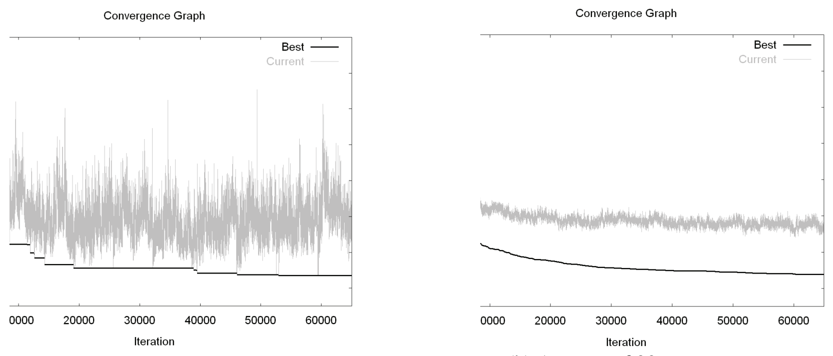
For a longer series of experiments (with the so called multi-run version), in (Balas et al, 2008) the authors considered the benchmark set proposed by Brucker and Thiele (Brucker and Thiele, 1996) containing a total of 17 instances (the so called BT set). As these instances do not define due-dates, they are taken to be 0. Thus, minimizing the maximum lateness is equivalent to minimizing the makespan. BT instances are divided in three groups depending on its size: small instances, t2-ps01 to t2-ps05, are 10×5, medium instances, t2-ps06 to t2-ps10, are 15×5, and large instances, t2-ps11 to t2-ps15, t2-pss12 and t2-pss13, are 20×5.

The time given to $TS - N_L^S$ (coded in C++) was similar to the time taken by SB-GLS as reported in (Balas et al, 2008). In order to establish this time, we have run the SB-GLS implementation from http://www.andrew.cmu.edu/ neils/tsp (coded in C) on our machine (Intel Core 2 Duo at 2.6GHz, gcc 3.4.4 compiler and Windows XP) and registered the time taken for each subfamily of instances. Then, we have parameterized $TS - N_L^S$ in such a way that it takes equivalent or less time for all instances on the same machine. In average, we have given $TS - N_L^S$ about 60% of the time taken by SB-GLS and in no case was the time given to $TS - N_L^S$ longer than the time taken by SB-GLS. The algorithm ISS was implemented in Allegro Common LISP 6.0 and run on a 0.9 Ghz machine, with runtimes between 40 and 3200 seconds, making it harder to establish equivalent runtimes.

5.1 Calibration of the TS algorithm

Calibration is often one of the most important steps to obtain good results. One of the main advantages of the algorithm proposed in Section 3 is that it does not require too much effort for parameter tuning, as $maxGlobalIter$ is calculated from the time limit and $maxImproveIter$ is fixed to $maxGlobalIter/Size(ESL)$. Therefore, we only have to analyze the size of the elite solution stack and the length of the tabu list. To do that, we have experimented across the largest instances of the BT set with $maxGlobalIter = 60000$: 30 runs were done for each instance and the average of the best solutions considered. Five different values have been tested for the stack size (0, 5, 10, 20 and 40). With little difference for the five values under consideration, the best results were obtained for size 10 (1412.86) and the worse ones, for size 40 (1418.43). Three different values were considered for the length of the tabu list (3, 8 and 16) with fixed length, as well as testing dynamic length. In this case, dynamic length was clearly the best option (1412.86 vs. 1442.71).

In order to further assess the chosen parameters, we have analyzed the convergence pattern of $TS - N_L^S$ algorithm across the iterations. Some results of these experiments are shown in Figure 3 (a) and (b). The first represents the evolution of $TS - N_L^S$ in a single run for instance $i305\_141$, while the latter represents the evolution averaged across 30 runs. Both illustrate a proper convergence pattern with improvement along the search process. We can also see that the generated initial solution is already quite good, as mentioned in Section 3. Finally, the evolution of the current solution displays a convergent but not monotone trajectory. Such gradual and nonmonotone improvement is characteristic of the general behavior of TS.



(a) One run (b) Average of 30 runs

Fig. 3 Evolution of the current and best solutions of $TS - N_L^S$ for the instance i305_141.

5.2 Relative Errors

The aim of the following experiments is to assess how close the algorithm comes to producing an optimal schedule, measured as percent error versus the optimal solution value. However, as far as we know, there are neither satisfactory lower bounds nor statistical estimates of the optimum for these instances, so we use the best-known solutions (BKS) instead. As source of BKS, we shall use the best values reported so far in the literature together with the values obtained with long runs (10^6 iterations, see below) of our method, as this often yields better solutions than those from the literature. The same BKS has been used to calculate relative errors for all three methods.

An added difficulty of dealing with maximum lateness is that objective values may be negative. To cope with this, when calculating relative errors, we add the minimum due date to lateness values in both numerator and denominator of ratios (since at least the lateness the job with minimum due date will not be negative). Notice that this transformation understates the percent error of heuristic solutions. However, since this is the case for the three methods considered, comparisons among them should not be affected.

Table 1 summarizes the results reported for SB-GLS (column 6), ISS (column 7), and our method $TS - N_L^S$ when it is run for an equivalent time to SB-GLS (columns 8 and 9 for the average and best result respectively across 30 runs). These results are obtained across the 960 instances and averaged for each group of 160 instances. The reason to include best and average values for $TS - N_L^S$ is that both SB-GLS and ISS are also approximate methods and it is not clear whether the reported values are actually sample means. Figure 4 shows the results given in Table 1 averaged for each one of the six families of instances defined by the same values of parameters τ and R.

Table 1: Average percent errors.

					SB-GLS	ISS	$TS - N_L^S$	
Group	τ	R	M	N			Avg.	Best.
i305	0.3	0.5	5	10	3.95	4.80	0.59	0.06
				20	10.19	21.50	2.33	0.81
			10	10	3.72	4.44	1.11	0.08
				20	11.38	19.91	2.27	0.53
			15	10	2.81	3.55	0.85	0.11

Table 1: Average percent errors (continued).

					SB-GLS	ISS	$TS-N_L^S$	
Group	τ	R	M	N			Avg.	Best.
				20	10.13	22.59	2.99	0.70
			20	10	2.71	3.02	0.77	0.12
				20	8.46	29.43	3.86	1.55
i315	0.3	1.5	5	10	2.06	1.96	0.83	0.04
				20	13.42	11.95	1.98	0.10
			10	10	1.56	2.23	0.59	0.08
				20	8.05	4.72	2.41	0.63
			15	10	2.71	2.27	0.94	0.02
				20	9.14	8.65	4.34	1.70
			20	10	1.60	2.24	1.29	0.10
				20	8.77	17.63	7.45	3.05
i325	0.3	2.5	5	10	1.36	2.48	0.30	0.40
				20	1.27	5.93	0.85	0.09
			10	10	1.20	2.36	1.94	0.18
				20	2.23	2.50	0.67	0.11
			15	10	1.62	0.86	1.13	0.01
				20	4.33	3.06	1.98	0.37
			20	10	1.08	0.63	0.86	0.05
				20	4.54	3.32	3.17	0.92
i605	0.6	0.5	5	10	5.41	4.44	0.79	0.16
				20	7.91	20.17	2.01	0.64
			10	10	3.12	4.31	1.04	0.16
				20	9.37	20.79	2.10	0.33
			15	10	2.81	3.93	0.87	0.03
				20	7.86	23.76	2.72	0.95
			20	10	2.95	4.11	1.38	0.29
				20	9.14	28.95	3.64	1.26
i615	0.6	1.5	5	10	5.22	4.60	0.64	0.00
				20	12.84	19.04	2.68	0.77
			10	10	2.54	3.57	0.81	0.08
				20	12.21	19.85	2.68	0.47
			15	10	2.34	2.65	1.47	0.29
				20	10.17	20.22	4.13	1.25
			20	10	2.33	2.71	1.09	0.13
				20	9.49	28.71	5.05	1.98
i625	0.6	2.5	5	10	3.93	4.13	0.64	0.05
				20	6.86	11.52	1.3	0.32
			10	10	1.87	2.72	0.98	0.13
				20	7.97	5.48	2.07	0.38
			15	10	2.27	2	1.09	0.27
				20	8.65	9.21	3.96	1.23
			20	10	2.05	1.44	0.9	0.12
				20	9.1	18.23	5.91	2.39
Overall average					9.35	5.56	1.99	0.53
Overall worst					48.04	47.84	13.47	6.46
Overall best					0.00	0.00	0.00	0.00

From the results it is clear that the proposed TS algorithm outperforms the other two methods in solution quality, yielding sensibly better results even in the worst case. Taking the mean values obtained with $TS-N_L^S$, the error reduction w.r.t. SB-GLS is in average 58.5% (32% for one family of instances and over 50% in the rest). Compared to ISS, $TS-N_L^S$ reduces the error 65.43% in average, and over 75% in three of the six families. The relevance of our proposal would even be greater if the results reported in (Balas et al, 2008) and (Oddi et al, 2009) were referred to best values; in this case, taking the best values produced by $TS-N_L^S$, the relative error reduction w.r.t. both methods would be more than 90%.

For a better insight into the behavior across the 960 instances, we have recorded the number of instances where $TS-N_L^S$ obtains better, equal or worse results than ISS and SB-

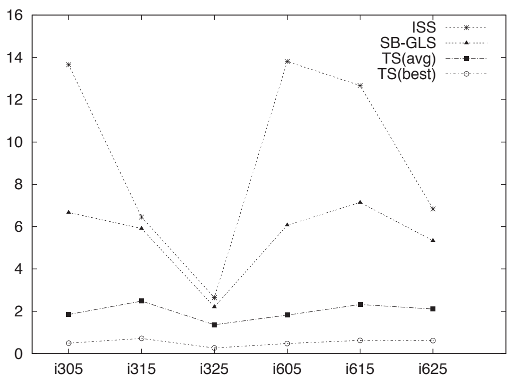


Fig. 4 Errors from each method averaged for the six families of instances

Table 2 Percent errors w.r.t. problem parameters for all algorithms.

	ISS	SB-GLS	Avg TS	Best TS
$n = 10$	2.98	2.63	0.95	0.12
$n = 20$	15.71	8.48	3.02	0.94
$m = 5$	9.38	6.20	1.25	0.29
$m = 10$	7.74	5.44	1.56	0.26
$m = 15$	8.56	5.40	2.21	0.58
$m = 20$	11.70	5.19	2.95	1.00
$R = 0.5$	13.73	6.37	1.83	0.49
$R = 1.5$	9.56	6.53	2.40	0.67
$R = 2.5$	4.74	3.77	1.73	0.44
$\tau = 0.3$	7.58	4.93	1.90	0.49
$\tau = 0.6$	11.11	6.18	2.08	0.57

GLS, with the average (best) values obtained by $TS - N_L^S$ improving the other algorithms in 87.4% (90.5%) and 80.0% (84.0%) of the instances respectively.

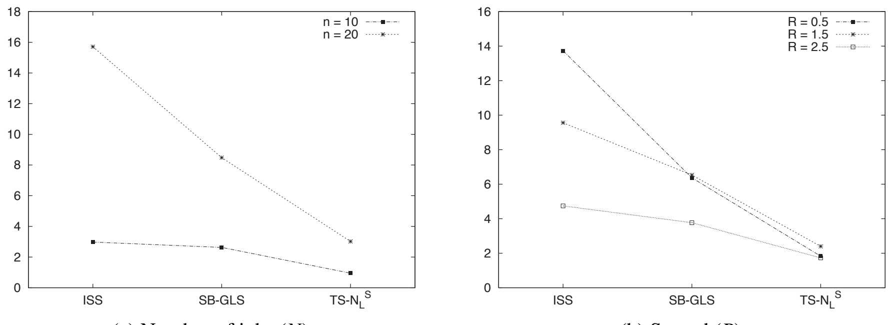
Additionally, we have computed 95% confidence intervals for the relative errors obtained with the three algorithms, and the interval upper bound for $TS - N_L^S$ (1,99 + 0,12= 2,11) is noticeably smaller than the interval lower bounds for ISS and SB-GLS (9.35 - 0.61 = 8.74 and 5.56 - 0.32 = 5.24 respectively). These results confirm that $TS - N_L^S$ is superior to ISS and SB-GLS both in quality and stability.

5.3 Sensitivity to Problem Parameters

Table 2 provides more detail on the effect of every parameter (n, m, R and τ) of the instances from (Ovacik and Uzsoy, 1994) on each algorithm performance, with average errors per algorithm for each possible parameter value. For the same reason as above, we also report data for the best solutions found with $TS - N_L^S$.

According to Table 2, parameter values have considerably less impact on $TS - N_L^S$ than on SB-GLS and ISS. This is also illustrated in Figure 5. Regarding the number of machines m, for $m = 5$ $TS - N_L^S$ obtains the best results and the greatest error reduction with respect to the other two methods: 87% less than ISS and 80% less than SB-GLS. For $TS - N_L^S$ the error increases with the number of machines, unlike for SB-GLS.

Regarding the number of jobs n, all algorithms obtain the smallest errors for $n = 10$. $TS - N_L^S$ obtains an average error reduction w.r.t. ISS of 74%, reaching 81% for the largest



(a) Number of jobs (N) (b) Spread (R)

Fig. 5 Interaction between parameters and algorithms, with the error w.r.t. BKS on the Y axis

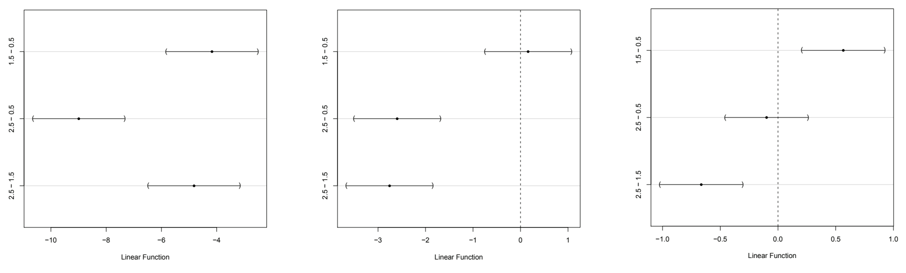
number of jobs. Compared to SB-GLS, the error reduction is 64%. As for tardiness τ, the smallest error rates are for $\tau = 0.3$, suggesting that the difficulty in problem instances increases with the number of expected tardy jobs. The greatest reduction in error rate is obtained by $TS - N_L^S$ when problem instances seem to be more difficult ($\tau = 0.6$): 81% w.r.t. ISS and 66% w.r.t. SB-GLS. Finally, regarding spread R, all methods obtain the smallest average error rates for $R = 2.5$, which suggests that problems are easier for the greatest width of the interval from which due dates are taken. The improvement of $TS - N_L^S$ for the remaining values of R is clear, with an error reduction of $75 - 87\%$ w.r.t. ISS and $63 - 71\%$ w.r.t. SB-GLS.

In all cases, the analysis takes as reference the average error value for $TS - N_L^S$, not the best performance value; obviously, the difference in performance would be greater if we were to compare these best values. Additionally, we have run several statistical tests to assess the interaction between parameters and algorithms: Shapiro-Wilk, Barlett, t-test and ANOVA tests have been applied; when normality or homoscedasticity hypothesis required to apply ANOVA did not hold, non-parametric Krukal-Wallis test has been used instead. When appropriate, Tukey tests have been applied to analyse multiple-pairwise interactions.

Since parameters τ and n can only take two different values, a t-test has been run for each method and parameter. The results show that differences exist between both values in all cases except in the $TS - N_L^S$ algorithm where there is no evidence to assume differences between values of τ (p-value=0.14).

Regarding the number of machines, m, for all algorithms we have used Kruskal-Wallis test as we cannot assume neither normality nor homoscedasticity. For ISS, the existence of differences for each value of m is rejected (ANOVA has given p-value=0.00 indicating that interactions exist). The influence of m in SB-GLS is also rejected, whereas for $TS - N_L^S$ both ANOVA and Kruskal-Wallis indicate that the probability that the value of m has no effect on the average error is practically null. Furthermore, the Tukey test shows that there are differences in average errors for each pair of values, except for 10 and 5 machines, where $TS - N_L^S$ obtains the best results and the greatest error reductions with respect to the other two methods.

Finally, for the spread value, R, the Kruskal-Wallis tests show that for all algorithms there are differences in the relative errors obtained for each group of instances. Tukey tests have also been run concluding that there are differences between average values of relative errors, with only two exceptions: the pair $2.5 - 0.5$ for $TS - N_L^S$ (where this method obtains the smallest average error rates) and $1.5 - 0.5$ for SB-GLS. Figure 6 shows the confidence



(a) ISS (b) SB-GLS (c) $TS-N_L^S$

Fig. 6 Confidence intervals of the pairwise analysis of parameter R for the three algorithms. All three pairs of R-values: $2.5-1.5$, $2.5-0.5$ and $1.5-0.5$ are considered in the Y-axis. X-axis shows the differences in errors

Table 3 Results from $TS-N_L^S$ for the short vs. long analysis

Problem family	Short Avg±SD	Short Time	Long Avg±SD	Long Time
i305	1.85±1.26	3.38	0.62±0.62	58.08
i315	2.48±2.74	2.38	0.71±1.07	41.12
i325	1.36±1.96	1.36	0.68±1.44	27.21
i605	1.82±1.18	3.69	0.61±0.64	60.10
i615	2.32±1.75	3.55	0.75±0.77	54.13
i625	2.11±2.17	2.67	0.56±0.71	43.51

intervals of the pairwise analysis of R for (a)ISS, (b)SB-GLS, and (c) $TS-N_L^S$ and complements the information given in Figure 5(b). As we can observe, the differences in average errors (X-axis) are much lower for $TS-N_L^S$ than they are for both ISS and SB-GLS. This is clear if we are aware of the ranges of X-axis in Figure 6(a), (b) and (c).

From this experimental analysis, we conclude that $TS-N_L^S$ is both efficient and robust with respect to parameter setting. For all combinations of the parameters, $TS-N_L^S$ is the most efficient and robust of the three algorithms evaluated. Also, ISS seems to be the most variable algorithm, although in the worst case it is SB-GLS the algorithm with the largest dependency on the parameters.

5.4 Further Analysis of Tabu Search

In a second set of experiments, $TS-N_L^S$ was run for a larger number of iterations, in order to evaluate its capability to improve with longer runtimes. Table 3 contains a summary of results obtained with the *short* runs, under equivalent conditions to SB-GLS, and *long* runs, with 10^6 iterations. The relative error values are averaged across the instances in each family and across the 30 runs of the tabu search algorithm. The standard deviation of the mean relative error and the time taken in average on a single run are also included in each case. Detailed results on the 960 instances both for short and long executions are given in Online Resource 1.

The standard deviation values in the *short* experiments already suggest that there is room for improvement if $TS-N_L^S$ is given longer runtime. This is confirmed with clearly better average and standard deviation values for the *long* runs, suggesting that the increase in number of iterations not only yields better but also more stable results.

Table 4 Results across BT instances. Srun stands for single run SB-GLS, while Mrun stands for multi-run SB-RGLS10

	SB-GLS		$TS - N_L^S$ Short			$TS - N_L^S$ Long		
Instance	Srun	Mrun	Best	Avg	Time	Best	Avg	Time
t2-ps01	815	**798***	**798***	**798.0***	0.20	**798***	**798.0***	65.02
t2-ps02	807	**784***	**784***	**784.0***	0.20	**784***	**784.0***	65.85
t2-ps03	771	**749***	**749***	749.7	0.21	**749***	**749.0***	67.66
t2-ps04	738	**730***	**730***	732.8	0.18	**730***	**730.0***	59.83
t2-ps05	693	693	**691***	692.9	0.20	**691***	**691.0***	68.67
t2-ps06	1027	1018	1026	1026.0	0.90	**1013**	1025.3	116.61
t2-ps07	1022	1003	**970***	972.3	0.84	**970***	**970.0***	110.94
t2-ps08	1013	975	966	971.7	0.86	**963**	963.2	111.00
t2-ps09	1101	**1060**	**1060**	**1060.0**	0.94	**1060**	**1060.0**	122.07
t2-ps10	1051	**1018***	**1018***	1019.3	0.96	**1018***	**1018.0***	124.69
t2-ps11	1572	1470	1451	1467.5	3.51	**1443**	1449.9	173.86
t2-ps12	1369	1305	**1274**	1306.6	3.40	**1274**	1278.3	168.74
t2-ps13	1439	1439	**1415**	1426.7	3.51	**1415**	**1415.0**	174.67
t2-ps14	1602	**1485**	1492	1496.2	4.27	1492	1492.0	213.03
t2-ps15	1542	1527	1504	1524.7	3.51	**1485**	1500.4	174.18
t2-pss12	1305	1290	1269	1280.8	3.59	**1259**	1268.3	178.54
t2-pss13	1409	1398	**1370**	1385.9	3.69	**1370**	1374.1	184.40

*: optimal value, bold: best value

We have also compared our algorithm with the multi-run version proposed in (Balas et al, 2008). This version uses a variant of SB-GLS termed SB-RGLS10 and implements a two-phase procedure: the first phase is the algorithm SB-GLS, and the second one is a modified version of SB-GLS, where the bottleneck machine is chosen at random from a uniform distribution and which is repeated 40 times, therefore being much more time-consuming than SB-GLS.

Table 4 summarizes the results of SB-GLS (Srun) and multi-run SB-RGLS10 (Mrun) versions, together with results of $TS - N_L^S$ with short and long runs. The times given to $TS - N_L^S$ in short and long runs are equivalent to the times taken by SB-GLS and multi-run SB-RGLS10 respectively, considering for the latter the longest among the experiments reported in (Balas et al, 2008). For the short runs, $TS - N_L^S$ was thus run for 9000 iterations for instances t2-ps01 to t2-ps05, 23000 iterations for instances t2-ps06 to t2-ps10, and 60000 for the remaining instances. For the long runs, $TS - N_L^S$ was run for 3000000 iterations.

Regarding the short runs, the average solutions reached by $TS - N_L^S$ are better than those reached by SB-GLS in all 17 instances. For the long runs, both methods reach the same solution in 6 instances—where the optimal solutions are obtained— $TS - N_L^S$ is better than multi-run SB-RGLS10 in 9 instances (including 6 of the 7 largest instances), and only in 2 instances is $TS - N_L^S$ worse (in average) than multi-run SB-RGLS10. Also, in 15 of the 17 instances the average value of the long run is better than the best value of the short run.

6 Conclusions and Future Work

We have proposed a method, $TS - N_L^S$, to solve the SDST-JSP with maximum lateness minimization. We have presented a series of experiments which show that $TS - N_L^S$ outperforms SB-GLS - one of the best methods in the literature, under equivalent running times. It also improves the results of ISS, although in this case we have to be aware of the difference in the running conditions. Additionally, we have shown the robustness of our method to changes in the parameters used to define problem instances from (Ovacik and Uzsoy, 1994). Finally, we have seen how the results can be further improved when $TS - N_L^S$ is given longer running time. Indeed, in our experiments, $TS - N_L^S$ has established new best-known solutions for 817 of the 960 instances in this difficult benchmark, which should serve as reference for future research.

We may conclude that $TS-N_L^S$ has proved to be a highly competitive method for the SDST-JSP with maximum lateness minimization. We believe the strongest points of $TS-N_L^S$ to be the powerful tabu search algorithm, already successful in other similar problems (Dell' Amico and Trubian, 1993; Nowicki and Smutnicki, 1996) as well as the neighborhood structure N_L^S. This structure has been specifically designed for this problem and is therefore very well adapted to its characteristics, in particular, to setup times (even those not fulfilling the triangular inequality) and the objective function (maximum lateness). In principle, N_L^S requires exploring a large number of moves. However, the theoretical results in Theorems 1 and 2 allow to discard a great number of candidate neighbors which either are not feasible or cannot improve the current schedule, so the number of actual neighbors in N_L^S can be managed efficiently. Additionally, the algorithm given in Section 4.1 is very efficient and accurate in estimating the maximum lateness of a new neighbor, giving $TS-N_L^S$ the chance of selecting one of the best neighbors. Given the specialization of the defined neighborhood, the algorithm would require a series of modifications in order to solve other versions of this problem with different characteristics or objective functions. In particular, a new disjunctive model would be required that accurately represents these characteristics and new formal results should be established to devise a new efficient neighborhood structure.

A first task for future work is to analyze the advantages and disadvantages of incorporating to our TS algorithm different strategies for handling long term memory, such as those used by two of the most competitive algorithms for classical JSP: path relinking of *i*-TSAB from (Nowicki and Smutnicki, 2005) or simulated annealing from (Zhang et al, 2008). In particular, it will be necessary to study its effectiveness when faced with the increase in complexity caused by setup times. We also intend to combine $TS-N_L^S$ with a genetic algorithm, expecting that the combined approach will further improve the solutions. Additionally, we plan to extend our approach to other variants or extensions of classical JSP which are closer to real environments and usually result harder to solve. For example the JSP with uncertain durations (González Rodríguez et al, 2010) or the JSP with other objective functions such as total flow time (Sierra and Varela, 2010) or weighted tardiness, as well as multiobjective problems.

Acknowledgements This work has been supported by the Spanish Ministry of Science and Education under research grant MEC-FEDER TIN2010-20976-C02-02 and by the Principality of Asturias under grant FICYT-BP07-109. The authors would also like to thank the Unidad de Consultoría Estadística from the Scientific and Technical Services of the University of Oviedo for their support in the statistical analysis of the experimental results.

References

Allahverdi A, Ng C, Cheng T, Kovalyov M (2008) A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187:985–1032

Armentano V, Filho M (2007) Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based grasp approach. European Journal of Operational Research 183:100–114

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

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

Balas E, Lancia G (1998) Job shop scheduling with deadlines. Journal of Combinatorial Optimization 1:329–353

Balas E, Simonetti N, Vazacopoulos A (2008) Job shop scheduling with setup times, deadlines and precedence constraints. Journal of Scheduling 11:253–262

Beck JC, Feng T, Watson JP (2010) Combining constraint programming and local search for job-shop scheduling. Informs Journal on Computing DOI: 10.1287/ijoc.1100.0388

Bilge U, Kiraç F, Kurtulan M, PekgÂÿn P (2004) A tabu search algorithm for parallel machine total tardiness problem. Computers & Operations Research 31:397–414

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

DeBontridder K (2005) Minimizing total weighted tardiness in a generalized job shop. Journal of Scheduling 8:479–496

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

Fleszar K, Charalambous C, Hindi KS (2011) A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times. Journal of Intelligent Manufacturing DOI: 10.1007/s10845-011-0522-8

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

González MA, Vela C, Varela R (2008) A new hybrid genetic algorithm for the job shop scheduling problem with setup times. In: Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-2008), AAAI Press, Sidney, pp 116–123

González MA, Vela CR, Varela R (2009) A tabu search algorithm to minimize lateness in scheduling problems with setup times. In: Proceedings of CAEPIA 2009, pp 115–124

González Rodríguez I, Vela CR, Puente J (2010) A genetic solution based on lexicographical goal programming for a multiobjective job shop with uncertainty. Journal of Intelligent Manufacturing 21 (1):65 – 73

Grabowski J, Wodecki M (2005) A very fast tabu search algorithm for job shop problem, Operations Research/Computer Science Interfaces Series, vol 30, Springer, US, pp 117–144

Graham R, Lawler E, Lenstra J, Rinnooy Kan A (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 5:287–326

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

Huang J, Suer G, Urs S (2011) Genetic algorithm for rotary machine scheduling with dependent processing times. Journal of Intelligent Manufacturing DOI: 10.1007/s10845-011-0521-9

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

Meeran S, Morshed M (2011) A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study. Journal of Intelligent Manufacturing DOI: 10.1007/s10845-011-0520-x

Mehta S, Uzsoy R (1998) Predictable scheduling of a job shop subject to breakdowns. IEEE Transactions on Robotics and Automation 14(3):365–378

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

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

Oddi A, Rasconi R, Cesta A, Smith S (2009) Iterative-sampling search for job shop scheduling with setup times. In: COPLAS 2009 Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems, pp 27–33

Ovacik I, Uzsoy R (1994) Exploiting shop floor status information to schedule complex job shops. Journal of Manufacturing Systems 13(2):73–84

Pinedo M (2008) Scheduling. Theory, Algorithms, and Systems. Springer, third edition

Serifoglu F, Ulusoy G (1999) Parallel machine scheduling with earliness and tardiness penalties. Computers & Operations Research 26:773–787

Sierra M, Varela R (2010) 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

Stecco G, Cordeau JF (2009) A tabu search heuristic for a sequence-dependent and time-dependent scheduling problem on a single machine. Journal of Scheduling 12:3–16

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

Uzsoy R, Velásquez J (2008) Heuristics for minimizing maximum lateness on a single machine with family-dependent set-up times. Computers and Operations Research 35:2018–2033

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

Vela CR, Varela R, González MA (2010) Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. Journal of Heuristics 16:139–165

Watson J, Howe A, Whitley L (2006) Deconstructing Nowicki and Smutnicki's i-TSAB tabu search algorithm for the job-shop scheduling problem. Computers and Operations Research 33:2623–2644

Wilbrecht J, Prescott W (1969) The influence of setup times on job shop performance. Management Science 16(4):391–401

Yan H, Xia Q, Zhu M, Liu X, Guo Z (2003) Integrated production planning and scheduling on automobile assembly lines. IIE Transactions 35(8):711–725

Ying KC, Lee ZJ, Lin SW (2011) Makespan minimization for scheduling unrelated parallel machines with setup times. Journal of Intelligent Manufacturing DOI: 10.1007/s10845-010-0483-3

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