J Sched (2009) 12: 3–16
DOI 10.1007/s10951-008-0068-6

A tabu search heuristic for a sequence-dependent and time-dependent scheduling problem on a single machine

Gabriella Stecco · Jean-François Cordeau · Elena Moretti

Received: 16 April 2007 / Accepted: 22 April 2008 / Published online: 12 July 2008

Abstract This paper introduces a tabu search heuristic for a production scheduling problem with sequence-dependent and time-dependent setup times on a single machine. The problem consists in scheduling a set of dependent jobs, where the transition between two jobs comprises an unrestricted setup that can be performed at any time, and a restricted setup that must be performed outside of a given time interval which repeats daily in the same position. The setup time between two jobs is thus a function of the completion time of the first job. The tabu search heuristic relies on shift and swap moves, and a surrogate objective function is used to speed-up the neighborhood evaluation. Computational experiments show that the proposed heuristic consistently finds better solutions in less computation time than a recent branch-and-cut algorithm. Furthermore, on instances where the branch-and-cut algorithm cannot find the optimal solution, the heuristic always identifies a better solution.

Keywords Single machine scheduling · Sequence-dependent · Time-dependent · Setup times · Tabu search

G. Stecco · E. Moretti
Department of Applied Mathematics, University Ca' Foscari of Venice, Dorsoduro n. 3825/E, 30123 Venice, Italy

G. Stecco
e-mail: gabriellastecco@unive.it

E. Moretti
e-mail: emoretti@unive.it

J.-F. Cordeau (✉)
Canada Research Chair in Logistics and Transportation and CIRRELT, HEC Montréal 3000 chemin de la Côte-Sainte-Catherine, Montréal H3T 2A7, Canada
e-mail: cordeau@crt.umontreal.ca

1 Introduction

This paper introduces a tabu search heuristic for a *Sequence- and Time-Dependent Scheduling Problem* (STDSP) on a single machine. In this problem the setup between two jobs is divided into two parts: one that can be performed at any time and another one that must be performed outside of a *forbidden interval* which repeats daily in the same position. This problem arises, for example, in situations where production can be continued but not initiated during certain parts of the day. This is often the case when workers with special qualifications are required to perform some of the setups. This problem is, in fact, inspired from the actual operation of a plastic container manufacturer where the transition between jobs may require two setups: a mould and a color setup. Night workers can perform only color setups, and mould setups must be performed during the day.

In our problem, the transition between two jobs thus involves two setups: an *unrestricted setup* that can be performed at any time, and a *restricted setup* that is required to take place outside of the forbidden interval. The processing of a job may continue during this interval. However, if a job is completed during the interval, only the unrestricted setup of the next job can be performed. If a job is completed before this interval and the restricted setup cannot be performed in full, it is possible to divide it into two parts: one that is performed before the forbidden interval and the other after the interval. In addition, the unrestricted setup can be performed either before or after the restricted setup, but it cannot be performed during the forbidden interval if the restricted setup is performed only in part before that interval. To calculate the setup time between two jobs it is thus necessary to know the completion time of the first one, which determines whether both setups can be performed consecutively without interruption, or if there is an idle time due to

the restricted setup. Hence, the total setup time between two jobs is a function of the time at which the first of these jobs is completed.

One can find in the production scheduling literature several studies addressing problems where the processing time of a job depends on its starting time in the schedule. The problem considered in this paper is more general as it deals with both sequence-dependent and time-dependent setup times (time-dependent setup times are in fact equivalent to time-dependent processing times because the dependent part of the setup time, in this case the idle time, can be added either to the processing time or to the setup time). Different types of functions defining the processing time of a job have been considered in the literature (see Alidaee and Womer 1999 and Cheng et al. 2004 for surveys). Recent research along these lines has focused on problems with linear or piecewise linear time-dependent processing times. Mosheiov (1995) and Cheng and Ding (2001) have shown that the problem with a single machine where the processing time is a step function is NP-complete. In particular, Jeng and Lin (2004) have proposed a pseudo-polynomial time dynamic programming algorithm and a branch-and-bound algorithm for the problem with two steps. Lahlou and Dauzère-Pérès (2006) have considered a single machine, a time horizon divided into time windows and job processing times that depend on the time window in which a job starts. Finally, Khammuang et al. (2007) have described an on-line scheduling problem with forbidden intervals during which a job can be processed but a new one cannot start. The authors discuss the relationship between this problem and the bin packing problem. However, none of these studies considers sequence-dependent setup times and they deal only with time-dependent processing times.

Two recent articles by Barketau et al. (2005) and by Leung et al. (2006) consider (sequence independent) setup times and time-dependent processing times. Barketau et al. (2005) have studied the problem of scheduling n jobs on a single machine. The jobs are processed in batches and each batch is preceded by a setup time S. The processing time of a job is a step function that depends on the time elapsed since the start of the processing of the batch to which the job belongs. The objective is the minimization of the makespan, $C_{\max}$. The authors have shown that the problem is NP-hard. Leung et al. (2006) have studied a similar problem with m parallel machines and with the sum of completion times, $\sum C_i$, as objective function. They have shown that the problem is NP-hard and they have described both exact and approximation algorithms.

The STDSP can also be formulated as a time-dependent asymmetric traveling salesman problem (TDTSP). The TDTSP is a TSP where the length of an arc is a function of the departure time from the origin of the arc (see Malandraki and Daskin 1992). Under this representation, every node represents a job and the length of an arc is the setup time between the two corresponding jobs. There exists another definition of the time-dependent TSP that should not be confused with that considered here. Indeed, Picard and Queyranne (1978) and some other authors have considered a special time-dependent TSP where the travel time between two nodes depends on the position (or rank) of the first node in the sequence and not on the departure time from the first node. Recently, Bigras et al. (2006) have studied scheduling problems on a single machine with sequence dependent setup times in which the objective is to minimize the makespan and the total tardiness. They have proposed a branch-and-bound algorithm and formulations inspired by the definition of the TDTSP given by Picard and Queyranne (1978).

For the general TDTSP, a mixed integer linear programming formulation was introduced by Malandraki and Daskin (1992). These authors have considered a step travel time function of the departure time from the first node and have studied some properties of the problem. They have also proposed a cutting plane algorithm as well as heuristics. Computational results were reported for a travel time function with two or three intervals. A restricted dynamic programming heuristic as a generalization of the nearest-neighbor heuristic was later proposed by Malandraki and Dial (1996) who reported results on randomly generated problems with two or three periods for each arc. More recently, Albiach et al. (2007) have described an exact method to solve the asymmetric traveling salesman problem with time-dependent costs and time windows by transforming it into an asymmetric generalized TSP and then into a graphical asymmetric TSP. They then apply an exact algorithm for the Mixed General Routing Problem. Computational results were reported on data sets adapted from ATSPTW instances. They have solved optimally instances with up 60 nodes. Ichoua et al. (2003) have proposed a model based on time-dependent travel speeds for the vehicle dispatching problem, and a parallel tabu search heuristic. In their algorithm the authors use a cross-exchange neighborhood (two segments of variable lengths are taken from two different routes and swapped) and an approximate neighborhood evaluation procedure to reduce computing times. Finally, a metaheuristic based on a multiple ant colony system was recently introduced by Donati et al. (2006) to solve the time-dependent VRP, while a genetic algorithm was presented by Haghani and Jung (2005) for the dynamic time-dependent VRP.

The STDSP considered in this paper was recently introduced by Stecco et al. (2008). The authors have introduced three mathematical formulations based on the TDTSP formulation of Malandraki and Daskin (1992). They have

also described valid inequalities which have been used within a branch-and-cut algorithm. This algorithm is able to solve some randomly generated instances with up to 50 jobs within reasonable computing times. However, some instances with only 20 jobs could not be solved within four hours of computing time.

The purpose of this paper is to introduce a fast tabu search heuristic capable of identifying high quality solutions in very short CPU times. Our computational experiments show that this heuristic is able to find in just a few minutes solutions that are equal to or better than those found by the branch-and-cut algorithm with much longer computing times.

The remainder of the paper is organized as follows. Section 2 describes the problem and provides a mathematical formulation. The tabu search algorithm is presented in Sect. 3. Finally, computational experiments are reported in Sect. 4, followed by the conclusions.

2 Problem description and formulation

In this section, we start by formally describing the STDSP. We then present a mathematical formulation which was introduced by (Stecco et al. 2008).

2.1 Notation

Let n be the number of jobs to be scheduled on a single machine over a given planning horizon. For notational convenience, we also introduce two dummy jobs, denoted by 0 and $n+1$. Job 0 represents the last job processed in the previous planning horizon. Let $N = \{0, 1, \ldots, n, n+1\}$, $P = \{0, \ldots, n\}$, and $S = \{1, \ldots, n+1\}$ be the sets of jobs to schedule, possible predecessors, and possible successors, respectively. Each job $j \in N \setminus \{0\}$ has a processing time p_j with $p_{n+1} = 0$. Preemption is not allowed. The setup between two jobs is divided into two parts: r_{ij} is the restricted setup time between jobs i and j, and u_{ij} is the unrestricted setup time. All jobs are ready to be processed at time t, which represents the beginning of the planning horizon. Let also d denote the length of a planning period (e.g., one day) and let $[a, b] \subset [0, d]$ denote the forbidden interval during which the restricted setup cannot be performed. Since the actual position of the forbidden interval within a period is not relevant, we assume, without loss of generality, that $[a, b]$ is of the form $[a, d]$, i.e., the end of the forbidden interval coincides with the end of each period.

Let t_j, for $j \in N$, be a real-valued decision variable representing the completion time of job j, and s_{ij} be the total setup time between jobs i and j. As explained, the total setup time between i and j is a function of the time at which job i is completed, hence, $s_{ij} = f(t_i)$ where $f : \mathbb{R}^+ \to \mathbb{R}^+$ is defined as follows:

$$f(t_i) := \begin{cases} r_{ij} + u_{ij} & \text{if } t_i \in [0, a - r_{ij}], \\ r_{ij} + u_{ij} + d - a & \\ \quad \text{if } t_i \in \big(a - r_{ij}, a - \min(r_{ij}, u_{ij})\big], & \\ r_{ij} + d - t_i & \text{if } t_i \in \big(a - \min(r_{ij}, u_{ij}), d - u_{ij}\big], \\ r_{ij} + u_{ij} & \text{if } t_i \in (d - u_{ij}, d]. \end{cases}$$

The total setup time between i and j is equal to the sum of r_{ij} and u_{ij} plus the idle time. If job i terminates during the first interval, $[0, a - r_{ij}]$, it is possible to perform the restricted setup before the forbidden interval and the idle time is then equal to zero. If job i ends during the second interval, $(a - r_{ij}, a - \min(r_{ij}, u_{ij})]$, it is impossible to fully perform the restricted setup before a. In this case, the unrestricted setup is performed before a and the restricted setup is divided into two parts: one before and the other one after the interval $[a, d]$. Then, $d - a$ is an idle time and is added to the setup time. If job i finishes during the third interval, $(a - \min(r_{ij}, u_{ij}), d - u_{ij}]$, one can make the unrestricted setup, wait until the end of $[a, d]$ and then perform the restricted setup. In this way the idle time is equal to the waiting time from the end of the unrestricted setup, if it was performed, to the end of the forbidden interval. Finally, if job i ends during the fourth interval, $(d - u_{ij}, d]$, it is again possible to perform both setups without interruptions, as in the first interval, with idle time equal to 0.

The function of the total setup time is illustrated in Fig. 1 for the interval $[0, d]$ and assuming $r_{ij} \geq u_{ij}$ and $d - u_{ij} \geq a - r_{ij}$. Note that if $r_{ij} < u_{ij}$ then the second interval $(a - r_{ij}, a - \min(r_{ij}, u_{ij})]$ does not exist. In addition, if $d - u_{ij} < a - r_{ij}$ $(d - a < u_{ij} - r_{ij})$ or if $r_{ij} = 0$ then the total setup time is constant.

Figure 2 shows an example of a production schedule with 3 jobs. One can observe that between jobs 1 and 2 there is an idle time because it is not possible to perform the restricted setup. There is also an idle time between jobs 2 and 3 because the restricted setup cannot be completely performed before the beginning of the forbidden interval.

2.2 A non-linear formulation

To formulate the problem we define for each job pair (i, j), with $i \in P$ and $j \in S$, a binary variable x_{ij} equal to 1 if and only if job j is processed immediately after job i, and 0 otherwise. The problem can be formulated as the following non-linear mixed integer programming problem:

$$\min t_{n+1} \tag{1}$$

subject to

$$\sum_{i \in P} x_{ij} = 1, \quad \forall j \in S, \tag{2}$$

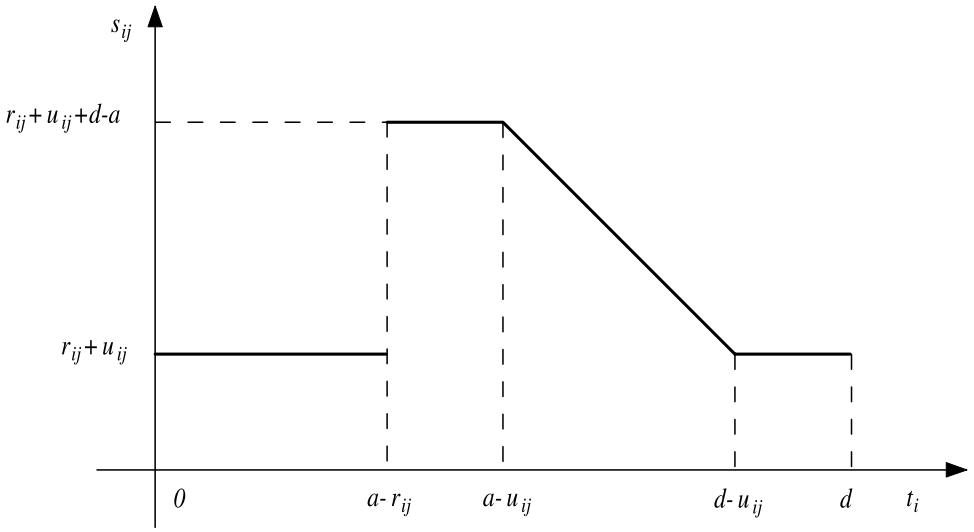

Fig. 1 Total setup time between jobs i and j

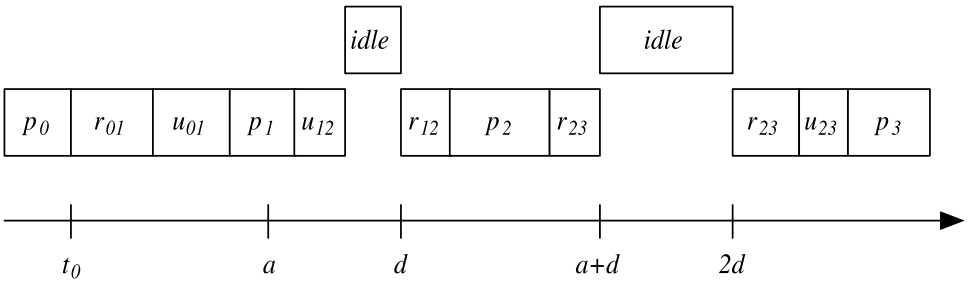

Fig. 2 Example of a production schedule

$$\sum_{j \in S} x_{ij} = 1, \quad \forall i \in P, \tag{3}$$

$$t_j - M x_{ij} \geq t_i + (s_{ij} + p_j) x_{ij} - M, \quad \forall i \in P, j \in S, \tag{4}$$

$$s_{ij} = f(t_i), \quad \forall i \in P, \ j \in S, \tag{5}$$

$$t_i \geq t, \quad \forall i \in N, \tag{6}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in P, \ j \in S. \tag{7}$$

In this model, the objective function (1) minimizes the completion time of the last job, $n + 1$. Constraints (2) and (3) ensure that each job is processed exactly once while constraints (4) ensure the consistency of the t_i variables. Constraints (5) define the total setup time as a function of the time at which job i finishes. Constraints (6) ensure that all jobs start after the beginning of the planning horizon, t. In this formulation, M is a large number used to linearize constraints (4). Any valid upper bound on the value of t_{n+1} can be used as the value of M.

As shown by Stecco et al. (2008), the above formulation can be linearized by reformulating the problem as a time-dependent TSP. The authors have also introduced an exact branch-and-cut algorithm that uses a mixed integer linear programming formulation.

3 Tabu search algorithm

In this section we describe the tabu search procedure used to solve the STDSP. Tabu search (see Glover and Laguna 1997 and Gendreau 2003) is one of the most popular techniques to find heuristic solutions to hard combinatorial optimization problems. Tabu search starts from an initial solution and moves at each iteration from the current solution to the best one in its neighborhood, even if this leads to a deterioration of the objective function value. To avoid cycling, attributes of recently visited solutions are declared tabu for a certain number of iterations. This process is repeated until a stopping criterion is satisfied.

For some problems, evaluating the cost of each solution in the neighborhood of the current solution may be very time consuming. Instead, one may identify a small set of promising solutions through a *surrogate objective function* and evaluate only the cost of these solutions. Most tabu search implementations use two types of memory: short term and long term memory. Short term memory is used to avoid cycling and trapping in local optima. At each iteration the move being performed is recorded and the reverse move is declared tabu for a certain number of iterations. The number of iterations during which a move is considered tabu is called the *tabu tenure*. The tabu status of a move can be revoked through an *aspiration criterion* if this move leads to

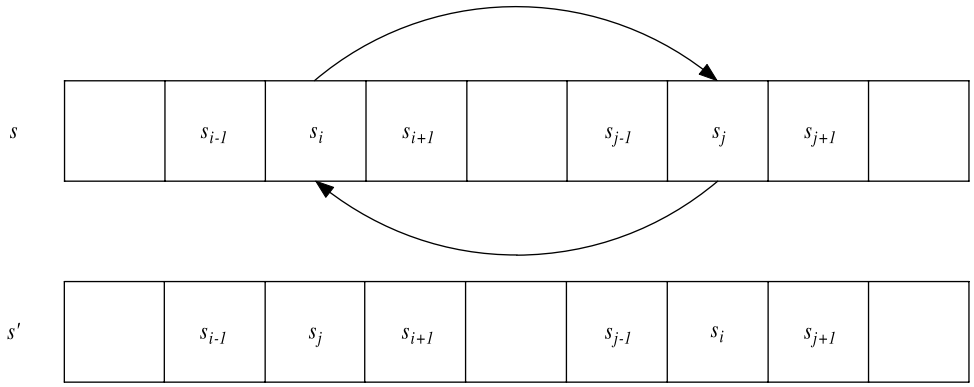

Fig. 3 Swap move between jobs s_i and s_j

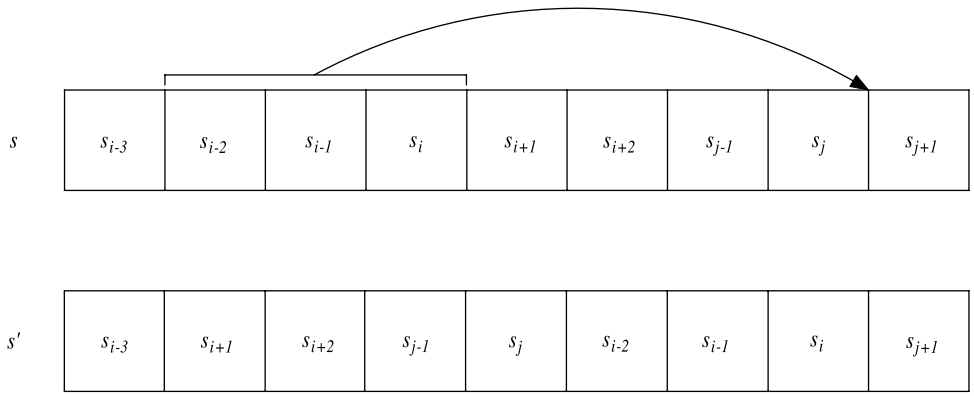

Fig. 4 Shift move between jobs s_i and s_j with $m = 3$

a new best solution. The long term memory is used to guide the search towards unexplored regions of the search space through a *diversification strategy*. In the next sections, we explain how these key aspects have been implemented in our tabu search heuristic.

3.1 Solution representation and initial solution

A schedule or solution s for the STDSP defines the processing order of jobs and it may be represented as a permutation of integers $s_0, s_1, s_2, \ldots, s_n, s_{n+1}$, where s_i denotes the job in position i (with $s_0 = 0$ and $s_{n+1} = n + 1$).

We use as an initial solution the permutation obtained by running a specialized nearest neighbor algorithm. The classical nearest neighborhood algorithm for the TSP is a greedy algorithm that, starting from a given city, chooses at each iteration the nearest city with respect to the one currently visited. In our problem, the greedy heuristic aims to minimize the sum of setup and idle times. At each iteration, it thus chooses a job j that minimizes the sum of setup and idle times between the current job and j. The procedure is repeated until all jobs are scheduled. The algorithm is applied n times by choosing each time a different starting job $i \in N \setminus \{0, n + 1\}$. The best solution found is then used as the starting solution for the tabu search algorithm.

3.2 Solution neighborhoods

In the scheduling and vehicle routing literature the *swap* and *shift* moves are probably the most popular exchanges considered to define solution neighborhoods. Given a solution s, the swap operator exchanges two jobs so that each job is located in the position previously occupied by the other one. Figure 3 illustrates a swap move involving two jobs, s_i and s_j.

The shift move chooses two positions i and j, with $i < j$ and a number m with $m \leq i$, which defines the number of jobs to move. It then relocates jobs $s_{i-m+1}, \ldots, s_i$ between s_j and s_{j+1}. In our algorithm we consider all the possible shift moves for every couple of positions i and j and for every value $m \leq i$. Figure 4 illustrates the shift move where $m = 3$ and three jobs are moved after jobs s_j.

The two neighborhoods defined by the swap and shift moves are evaluated separately and the best moves from each neighborhood are compared to choose the one to be performed. In case of equality, the shift move is selected.

3.3 Tabu status and tabu tenures

In this study we have considered both a fixed tabu tenure θ and random tabu tenures generated in the interval $[\mu, \nu]$ at each iteration. When relocating a group $s_{i-m+1}, \ldots, s_i$ after

job s_j we declare tabu moving job s_j to position j as well as moving job s_i to position j. A shift move is then considered tabu if moving the last job in the group, s_i, to position j is tabu. When a swap move is performed, moving job s_i to position i and moving job s_j to position j are both declared tabu. A swap move is considered tabu if one of the two associated exchanges is tabu.

3.4 Surrogate and auxiliary objectives

Evaluating the impact of a move on the objective function is costly in terms of computing time since it requires recalculating the completion time of every job that follows the first position affected by this move. This requires $O(n)$ operations which must be executed for every candidate move (there are $O(n^2)$ possible swap moves and $O(n^3)$ possible shift moves). To reduce the computational effort, we first perform an approximate evaluation of the neighborhood of the current solution s by using a surrogate objective function which is less demanding. This first step allows the identification of a set of promising candidates which are then evaluated exactly by using the true objective function.

Since the value of a solution is given by the sum of setup and idle times, for every solution s' in the neighborhood of s one can easy calculate the value of the setup time ($setup(s')$) knowing the setup time of s ($setup(s)$). In our implementation of swap and shift moves, position i is always before position j ($i < j$). In addition, the new solution s' coincides with s up to job s_{i-1} for the swap and up to s_{i-m} for the shift move. It is thus computationally easy to calculate the setup time of a solution. For every solution s we also store for every position i the value of the total idle time up to job s_i ($idle(s_i)$). The surrogate objective function for the swap move ($\tilde{z}(s)$) is

$$\tilde{z}(s') = setup(s') + idle(s_{i-1}),$$

where $idle(s_{i-1})$ is the idle time up to job s_{i-1}. For the shift move the surrogate objective function is

$$\tilde{z}(s') = setup(s') + idle(s_{i-m}).$$

Calculating the setup time of a solution s', knowing the setup time of s, is less demanding than calculating the real objective function. We then sum a part of the idle time that is certainly included in the new solution to better estimate the cost of the solution.

Each solution in the neighborhood of s with a value of the surrogate objective function less than or equal to α is then evaluated with the true objective function. If during the search the set of promising solutions is empty because no solution has a surrogate objective function value smaller than or equal to α, then the neighborhood is completely evaluated using the true objective function.

Another function, called an *auxiliary objective function*, is used to orient the search when the process is in a plateau where there are multiple solutions with the same objective value. If the best solution in the neighborhood of s has the same value as s and there are in the neighborhood more that one solution with this value we chose the move according to the following auxiliary objective function:

$$z'(s') = setup(s'),$$

i.e., the setup time for the solution s'. From the set of solutions minimizing the true objective function, we thus choose the one whose auxiliary objective function value is minimal. We prefer a solution with minimum setup time instead of minimum idle time because the setup time usually has a larger impact on the objective function value.

3.5 Diversification strategies

To diversify the search we use two strategies that operate on different levels and with a different intensity. A light diversification is based only on frequency memory. If the search cannot improve the current solution for β iterations, the frequency diversification is applied. After reducing the neighborhood using the surrogate objective function, each solution in the set of promising solutions is valued by adding to the true objective function a small penalty term. For every solution s' the frequency penalty $p(s')$ is equal to the sum, on every position i, of the number of iterations during which job s_i has been in position i, divided by the total number of iterations executed. The new value for the solution s' is then equal to

$$\hat{z}(s') = z(s') + f \cdot p(s') \cdot z(s')/\sqrt{n},$$

where f is a parameter to calibrate the intensity of the diversification. This type of function was first suggested by Taillard (1993) in the context of the vehicle routing problem. The penalty is scaled by a factor $z(s')/\sqrt{n}$ to account for instance size and for the magnitude of the objective function value. This type of diversification is used to drive the search towards less explored regions of the search space but, since the penalty is applied after the reduction of the neighborhood, the diversification has only a light impact. This diversification is also applied if the search is in a plateau and the algorithm finds for σ consecutive iterations a solution with the same objective function value. We have observed that in large instances there exist a large number of solutions with the same objective function value. When the algorithm is trapped in such a plateau it is difficult to escape and we prefer to apply the light diversification early instead of waiting until β iterations without improvement have passed.

A stronger diversification (called a setup diversification) is applied if the search does not improve the best solution

for γ iterations and the current solution for δ iterations. Usually, a strong diversification is applied if the search does not improve the best solution for some iterations. We use two parameters, γ and δ, to avoid jumping to a remote solution before we have finished exploring the current region. The strong diversification is applied also after δ iterations without improvement in the current solution. In the strong diversification, the reduced neighborhood is valued using a different objective function. The new objective function minimizes the setup time of s'. We add to this value a small term that includes the frequency penalty described above. The new value for s' is

$$\hat{z}(s') = setup(s') + f \cdot p(s') \cdot setup(s')/\sqrt{n}.$$

This diversification has a stronger impact because each solution (after reducing the neighborhood) is valued not with the real objective function but with only the setup time. The aim is to create a solution with minimum setup time but more idle time. The value of this solution is generally very far from the best solution but the search usually returns in a few iterations to solutions with a similar cost.

3.6 Aspiration and stopping criteria

Throughout the search, we use a classical aspiration criterion which revokes the tabu status of a move if it leads to a better solution than the best solution found so far.

The tabu search stops after a fixed number of iterations equal to 200 times the number of jobs to schedule. In the computational experiments, we have considered instances with between 5 and 50 jobs. The search thus performs between 1000 and 10,000 iterations.

4 Computational experiments

In this section we present the characteristics of the test instances used in our computational experiments. This is followed by a discussion of parameter calibration in Sect. 4.1 and by the computational results in Sect. 4.2. The tabu search algorithm was implemented in C and it was run on a 3.0 Hz Pentium 4 computer with 1 GB of memory.

4.1 Test instances

Our tabu heuristic was tested on the instances introduced by Stecco et al. (2008) and which are available on the following website: http://www.hec.ca/chairelogistique/data/stdsp. All of these instances were randomly generated according to the following parameters. The length d of a period is equal to 1440, i.e., the duration of one day expressed in minutes. The instances are divided into four sets whose characteristics are summarized in Table 1. The second column (a) indicates the time corresponding to the beginning of the forbidden interval in the first period of the planning horizon (this interval repeats in the same position in every subsequent period). Values for the restricted and unrestricted setup times as well as for the processing times were generated from a uniform distribution on the interval from 0 to the maximum values reported in the third, fourth and fifth columns, respectively.

Table 1 Characteristics of the four data sets

Instances	a	r_{ij}	u_{ij}	p_j
A1-1/A1-10	1380	30	30	60
A2-1/A2-10	1380	30	30	120
A3-1/A3-10	1380	30	30	180
B1-1/B1-10	1320	60	60	120
B2-1/B2-10	1320	60	60	240
B3-1/B3-10	1320	60	60	360
C1-1/C1-10	1200	120	120	240
C2-1/C2-10	1200	120	120	480
C3-1/C3-10	1200	120	120	720
D1-1/D1-10	960	240	240	480
D2-1/D2-10	960	240	240	960
D3-1/D3-10	960	240	240	1440

Table 2 Parameters used in the tabu search algorithm

θ	tabu tenure
α	factor used to define the set of promising solutions
β	number of iterations without improving the current solution before the frequency diversification is applied
γ	number of iterations without improving the best solution before the setup diversification is applied
δ	number of iterations without improving the current solution before the setup diversification is applied
σ	number of iterations allowed with the same objective value
f	factor used to adjust the intensity of the diversification

4.2 Parameter calibration

The behavior of the tabu search heuristic is controlled through a set of seven parameters which are listed in Table 2. A preliminary version of the algorithm was first tested on all instances to identify an appropriate range of values for each parameter. We have then set $\theta = (2\sqrt{n} + n)/2$, $f = 0.1$, $\alpha = z(s) + 1\%$, $\beta = n/4$, $\gamma = 9n$, $\delta = n/8$, and $\sigma = 5$. We next performed a sensitivity analysis on each of the seven parameters while keeping the other ones unchanged. The following order was used: $\theta, \sigma, \beta, \delta, \gamma, f$, and α. For the sensitivity analyses, we have used as test instances those of the fourth group (D) since they are the most difficult ones to solve.

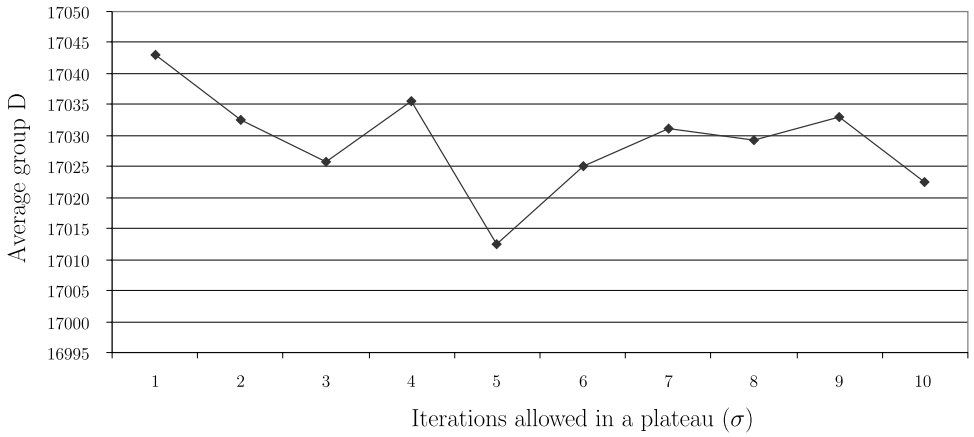

Fig. 5 Impact of parameter σ on average solution cost

4.2.1 Parameter θ

Having fixed the value of the other parameters as explained before, we ran tests using different values of the tabu tenure θ. The initial value $\theta = (2\sqrt{n} + n)/2$ appeared to yield good results for all instances, but slightly better results were obtained by setting $\theta = [7.5 \ln n]$, where $[x]$ is the integer nearest to x. Additional experiments performed with randomly chosen tabu tenures did not improve the results.

4.2.2 Parameter σ

After fixing $\theta = [7.5 \ln n]$ we ran tests to set the value of σ which controls the number of iterations allowed with the same objective value before the frequency-based diversification is applied. We considered values for σ in the interval $[1, 10]$ with increments of 1. Using a small value for σ seems to produce poor results because the diversification is applied too often. Similarly, using a too large value makes the algorithm spend too many iterations on a plateau before trying to escape. We have found that the best results were obtained with $\sigma = 5$ for most instances, regardless of their size. Figure 5 shows the average solution cost for different values of σ on the D instances.

4.2.3 Parameter β

Parameter β limits the number of iterations without improving the current solution. If during the search the algorithm does not improve the current solution for β iterations then the diversification based on frequency is applied. We ran tests with values in the interval $[1, 15]$ with increments of 1 for every instance. This parameter is sensible to the number of jobs to schedule and the best setting appears to be $\beta = n/4$.

4.2.4 Parameter δ

The value of δ is the number of iterations allowed without improving the current solution before the setup diversification is applied. This parameter is of course connected with β. We considered values in the interval $[n/8, n/4]$ with increments of $1/n$ and the best results were obtained with $\delta = n/8$.

4.2.5 Parameter γ

Parameter γ is also used to decide when the setup diversification has to be applied. If the algorithm cannot improve the best solution for γ iterations then the setup diversification is applied. This parameter is connected with the number of jobs to be scheduled and we ran tests using different values in the interval $[5n, 15n]$ with increments of n. The algorithm is rather insensitive to this parameter but the best results were obtained when γ was chosen in the interval $[9n, 11n]$ with $\gamma = 9n$ being the most appropriate.

4.2.6 Parameter f

The intensity of the diversification is controlled through parameter f. We have tested the algorithm for values in the interval $[0.01, 0.2]$ with increments of 0.01. As expected, using a too small value for f does not produce the desired diversification and the search is restricted to a small subset of the search space. Figure 6 reports the average solution cost for the D instances with different values of parameter f. One can see that values in the range $[0.1, 0.2]$ yielded good results, with 0.1 being the most appropriate value.

4.2.7 Parameter α

Lastly, we have varied the value of the parameter α which is used to reduce the neighborhood by considering only solutions for which the surrogate objective function value is below that parameter. We considered values for α in the range $[1.005z(s), 1.015z(s)]$ where $z(s)$ is the value of the current solution s, with increments of $0.001z(s)$. Our results showed that the results are comparable for most values in the interval with a slightly better performance for $\alpha = 1.01z(s)$.

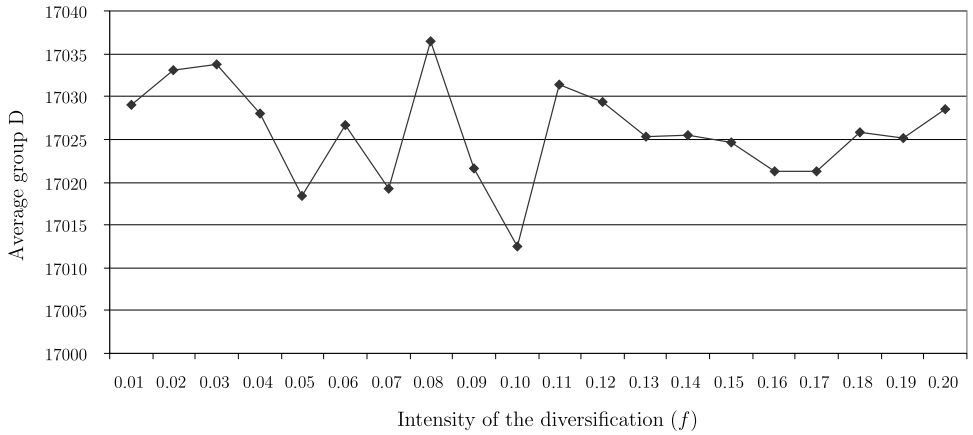

Fig. 6 Impact of parameter f on average solution cost

Table 3 Results with and without the surrogate objective function

Inst	With surrogate		Without surrogate		Change (%)	
	$z(s)$	CPU	$z(s)$	CPU	$z(s)$	CPU
A	2691.43	12.451	2691.43	75.071	0.000	682.466
B	4418.37	9.930	4418.26	77.048	0.000	657.409
C	8182.07	14.488	8177.96	80.561	−0.018	473.470
D	17012.43	32.706	17017.63	84.286	0.033	191.474

Table 4 Results with and without the diversification strategies

Inst	With diversification			Without diversification			Change (%)		
	$z(s)$	$z(s)-f$	CPU	$z(s)$	$z(s)-f$	CPU	$z(s)$	$z(s)-f$	CPU
A	2691.43	968.10	12.451	2691.86	968.53	12.631	0.012	0.037	−1.297
B	4418.37	1143.70	9.930	4417.20	1142.53	10.732	−0.023	−0.075	−2.230
C	8182.07	1643.04	14.488	8185.16	1646.13	14.259	0.042	0.151	−2.681
D	17012.43	2975.63	32.706	17045.36	3008.56	32.030	0.165	0.947	2.486

4.2.8 Impact of surrogate objective function and diversification

Table 3 reports the average solution cost and CPU time for every group of instances with and without using the surrogate objective function. One can see that on average the quality of the solutions is almost the same but the increase in CPU time goes from 682.466% for group A to 191.474% for group D.

Similarly, Table 4 reports the average solution cost and CPU time with and without the diversification strategies. The columns $z(s) - f$ report the results without the fixed costs. These results better highlight the impact of the diversification strategies. The impact is also more significant for the instances of group D which are the most difficult to solve. Finally, the diversification strategies do not have a big impact on CPU time.

4.3 Test results

In Tables 5, 6, 7, and 8 we report the results obtained by setting $\theta = [7.5 \ln n]$ $f = 0.1$, $\alpha = 1.01 z(s)$, $\beta = n/4$, $\gamma = 9n$, $\delta = n/8$, and $\sigma = 5$. The column headings are defined as follows:

- n, number of jobs to schedule;
- FIXED, value of the fixed cost for the instance, i.e., $\sum_{j=1}^{n} p_j$;
- LB, optimal solution found using the branch-and-cut algorithm of Stecco et al. (2008) or best lower bound (with 4 hours of computing time) if the instance was not solved to optimality (the symbol '*' indicates that the instance was solved to optimality). The branch-and-cut was implemented in C++ using ILOG Concert 1.3 and CPLEX 9.1.0 and was run on the same machine as the tabu search;

Table 5 Results for the first set of instances

Inst	n	FIXED	LB	UB	BKS	$z(s)$	$\%_{LB}$	$\%_{\mathrm{BKS}}$	CPU
A1-1	5	206.00	*1390.00	*1390.00	1390.00	1390.00	0.000	0.000	0.000
A1-2	10	313.00	*930.00	*930.00	930.00	930.00	0.000	0.000	0.050
A1-3	15	407.00	*1482.00	*1482.00	1482.00	1482.00	0.000	0.000	0.220
A1-4	20	665.00	*1571.00	*1571.00	1571.00	1571.00	0.000	0.000	0.480
A1-5	25	793.00	*1240.00	*1240.00	1240.00	1240.00	0.000	0.000	0.810
A1-6	30	761.00	*1533.00	*1533.00	1533.00	1533.00	0.000	0.000	1.920
A1-7	35	1005.00	*2395.00	*2395.00	2395.00	2395.00	0.000	0.000	4.050
A1-8	40	1273.00	*2010.00	*2010.00	2010.00	2010.00	0.000	0.000	6.340
A1-9	45	1374.00	*2690.00	*2690.00	2690.00	2690.00	0.000	0.000	13.460
A1-10	50	1622.00	2534.05	2553.00	2538.00	2538.00	0.156	0.000	17.130
A2-1	5	140.00	*1186.00	*1186.00	1186.00	1186.00	0.000	0.000	0.000
A2-2	10	661.00	*2171.00	*2171.00	2171.00	2171.00	0.000	0.000	0.060
A2-3	15	910.00	*2044.00	*2044.00	2044.00	2044.00	0.000	0.000	0.200
A2-4	20	1300.00	*2777.00	*2777.00	2777.00	2777.00	0.000	0.000	0.600
A2-5	25	1575.00	*3052.00	*3052.00	3052.00	3052.00	0.000	0.000	1.870
A2-6	30	2077.00	*2741.00	*2741.00	2741.00	2741.00	0.000	0.000	2.620
A2-7	35	2453.00	*3220.00	*3220.00	3220.00	3220.00	0.000	0.000	5.530
A2-8	40	2134.00	*2532.00	*2532.00	2532.00	2532.00	0.000	0.000	7.210
A2-9	45	2947.00	4393.18	4416.00	4397.00	4397.00	0.087	0.000	32.280
A2-10	50	2880.00	*3878.00	*3878.00	3878.00	3878.00	0.000	0.000	33.620
A3-1	5	592.00	*824.00	*824.00	824.00	824.00	0.000	0.000	0.000
A3-2	10	1158.00	*2007.00	*2007.00	2007.00	2007.00	0.000	0.000	0.050
A3-3	15	1441.00	*1869.00	*1869.00	1869.00	1869.00	0.000	0.000	0.160
A3-4	20	1699.00	*2150.00	*2150.00	2150.00	2150.00	0.000	0.000	0.490
A3-5	25	2583.00	*3715.00	*3715.00	3715.00	3715.00	0.000	0.000	1.780
A3-6	30	2783.00	*3551.00	*3551.00	3551.00	3551.00	0.000	0.000	4.290
A3-7	35	3234.00	*3897.00	*3897.00	3897.00	3897.00	0.000	0.000	9.940
A3-8	40	3879.00	*5555.00	*5555.00	5555.00	5555.00	0.000	0.000	38.700
A3-9	45	3744.00	*4975.00	*4975.00	4975.00	4975.00	0.000	0.000	44.400
A3-10	50	5091.00	6418.24	6449.00	6422.00	6423.00	0.074	0.016	145.260
Avg.		1723.33	2691.02	2693.43	2691.40	2691.43	0.011	0.001	12.451

- UB, optimal solution obtained using the branch-and-cut algorithm of Stecco et al. (2008) or best upper bound (with 4 hours of computing time) if the instance was not solved to optimality (as above, the symbol '*' indicates that the instance was solved to optimality);
- BKS, value of the best known solution (found during our sensitivity analyses);
- $z(s)$, result obtained by running the algorithm with the final parameter set;
- $\%_{\mathrm{LB}}$, gap between $z(s)$ and LB $((z(s) - \mathrm{LB})/\mathrm{LB} \times 100)$;
- $\%_{\mathrm{BKS}}$, gap between $z(s)$ and BKS $((z(s) - \mathrm{BKS})/\mathrm{BKS} \times 100)$;
- CPU, computing time in seconds.

The last row in each table indicates the average for each column. From the results, one can see that for the instances of group A (Table 5) the tabu search can find in all cases the optimal solution when the optimal solution is known, and for the three instances for which the branch-and-cut did not find the optimal value, the solution found by the tabu search is very close to the lower bound. The CPU time is on average around 5 seconds. For group B (Table 6), all the instances that were solved to optimality by the branch-and-cut can also be solved to optimality by the tabu search. For the other instances one can observe that the gap with the lower bound is small. The instances of group C (Table 7) and D (Table 8) are also solved to optimality if they were solved to

Table 6 Results for the second set of instances

Inst	n	FIXED	LB	UB	BKS	$z(s)$	$\%_{LB}$	$\%_{BKS}$	CPU
B1-1	5	182.00	*1234.00	*1234.00	1234.00	1234.00	0.000	0.000	0.000
B1-2	10	689.00	*1785.00	*1785.00	1785.00	1785.00	0.000	0.000	0.060
B1-3	15	918.00	*2154.00	*2154.00	2154.00	2154.00	0.000	0.000	0.170
B1-4	20	976.00	*1924.00	*1924.00	1924.00	1924.00	0.000	0.000	0.460
B1-5	25	1347.00	*2130.00	*2130.00	2130.00	2130.00	0.000	0.000	0.970
B1-6	30	1867.00	3346.00	3390.00	3365.00	3365.00	0.568	0.000	2.460
B1-7	35	2550.00	3160.32	3211.00	3170.00	3170.00	0.306	0.000	4.050
B1-8	40	2625.00	3971.42	4051.00	3981.00	3981.00	0.241	0.000	6.580
B1-9	45	3124.00	4688.09	4801.00	4713.00	4729.00	0.873	0.339	16.270
B1-10	50	3121.00	4619.00	4771.00	4659.00	4685.00	1.429	0.558	25.820
B2-1	5	445.00	*848.00	*848.00	848.00	848.00	0.000	0.000	0.000
B2-2	10	1373.00	*2071.00	*2071.00	2071.00	2071.00	0.000	0.000	0.060
B2-3	15	1384.00	*2186.00	*2186.00	2186.00	2186.00	0.000	0.000	0.160
B2-4	20	2251.00	*2904.00	*2904.00	2904.00	2904.00	0.000	0.000	0.480
B2-5	25	3134.00	*4681.00	*4681.00	4681.00	4681.00	0.000	0.000	1.530
B2-6	30	3042.00	*4460.00	*4460.00	4460.00	4460.00	0.000	0.000	2.700
B2-7	35	4021.00	5476.18	5505.00	5489.00	5494.00	0.325	0.091	5.860
B2-8	40	4725.00	6145.02	6224.00	6158.00	6169.00	0.390	0.179	12.660
B2-9	45	4909.00	6469.03	6530.00	6472.00	6480.00	0.170	0.124	19.650
B2-10	50	5701.00	7002.00	7114.00	7008.00	7028.00	0.371	0.285	35.180
B3-1	5	1003.00	*1659.00	*1659.00	1659.00	1659.00	0.000	0.000	0.000
B3-2	10	2273.00	*2792.00	*2792.00	2792.00	2792.00	0.000	0.000	0.050
B3-3	15	3110.00	*3693.00	*3693.00	3693.00	3693.00	0.000	0.000	0.170
B3-4	20	4330.00	*5512.00	*5512.00	5512.00	5512.00	0.000	0.000	0.710
B3-5	25	4579.00	*5762.00	*5762.00	5762.00	5762.00	0.000	0.000	1.810
B3-6	30	4958.00	*6558.00	*6558.00	6558.00	6558.00	0.000	0.000	4.860
B3-7	35	5537.00	7173.10	7192.00	7176.00	7176.00	0.040	0.000	10.100
B3-8	40	7368.00	8786.98	8842.00	8806.00	8813.00	0.296	0.079	23.360
B3-9	45	8356.00	9401.15	9462.00	9413.00	9416.00	0.158	0.032	44.410
B3-10	50	8342.00	9679.00	9813.00	9689.00	9692.00	0.134	0.031	77.310
Avg.		3274.67	4409.01	4441.97	4415.07	4418.37	0.177	0.057	9.930

optimality by the branch-and-cut. The other instances have a gap with the lower bound that increases with respect to the gap of group B, especially for the instances in C1. Nevertheless, if we compare the result of the algorithm with the best known solutions the gap is less than 1%. As one can expect the gap between the solutions found by the algorithm and the lower bound increases, especially for the D1 instances. However, if we compare the results with the best known solutions the gap remains at most around 1%.

One can also note that the CPU time increases on average from group A to group D, since the instances become more difficult and the number of solutions solved to optimality decreases. These computing times remain very small when compared with those of the branch-and-cut algorithm which was given a maximum of four hours for each instance.

Furthermore, a comparison with the upper bounds produced by the branch-and-cut algorithm shows that on all instances for which the branch-and-cut algorithm could not find an optimal solution within the time limit, the heuristic has identified a better solution. It thus seems fair to conclude that when both algorithms can solve an instance to optimality, the tabu search heuristic is faster, while for instances that cannot be solved optimally by the branch-and-cut, the heuristic produces a better solution in less than three minutes of computing time (compared to four hours for the former algorithm).

Table 7 Results for the third set of instances

Inst	n	FIXED	LB	UB	BKS	$z(s)$	$\%_{LB}$	$\%_{BKS}$	CPU
C1-1	5	635.00	*1677.00	*1677.00	1677.00	1677.00	0.000	0.000	0.000
C1-2	10	1690.00	*3113.00	*3113.00	3113.00	3113.00	0.000	0.000	0.070
C1-3	15	2190.00	*4434.00	*4434.00	4434.00	4434.00	0.000	0.000	0.340
C1-4	20	1924.00	*3430.00	*3430.00	3430.00	3430.00	0.000	0.000	0.570
C1-5	25	3276.00	5346.55	5433.00	5414.00	5414.00	1.262	0.000	1.740
C1-6	30	4001.00	5284.00	5397.00	5310.00	5316.00	0.606	0.113	2.530
C1-7	35	4294.00	5711.46	6103.00	5879.00	5915.00	3.564	0.612	11.100
C1-8	40	4940.00	6374.00	6774.00	6445.00	6506.00	2.071	0.946	12.220
C1-9	45	5112.00	6408.00	6857.00	6535.00	6554.00	2.278	0.291	18.370
C1-10	50	5982.00	8709.86	9401.00	8871.00	8910.00	2.298	0.440	54.040
C2-1	5	960.00	*2000.00	*2000.00	2000.00	2000.00	0.000	0.000	0.000
C2-2	10	2143.00	*4075.00	*4075.00	4075.00	4075.00	0.000	0.000	0.060
C2-3	15	3447.00	*5018.00	*5018.00	5018.00	5018.00	0.000	0.000	0.220
C2-4	20	5011.00	*5867.00	*5867.00	5867.00	5867.00	0.000	0.000	0.610
C2-5	25	5835.00	*8129.00	*8129.00	8129.00	8129.00	0.000	0.000	1.690
C2-6	30	7383.00	8689.33	8795.00	8732.00	8732.00	0.491	0.000	3.530
C2-7	35	9243.00	10326.00	10518.00	10388.00	10388.00	0.600	0.000	7.820
C2-8	40	9229.00	10911.50	11144.00	10937.00	10988.00	0.701	0.466	14.970
C2-9	45	9426.00	10795.00	11155.00	10858.00	10879.00	0.778	0.193	21.020
C2-10	50	12350.00	14479.20	14956.00	14567.00	14633.00	1.062	0.453	67.260
C3-1	5	1834.00	*2768.00	*2768.00	2768.00	2768.00	0.000	0.000	0.000
C3-2	10	3380.00	*4044.00	*4044.00	4044.00	4044.00	0.000	0.000	0.070
C3-3	15	5665.00	*7183.00	*7183.00	7183.00	7183.00	0.000	0.000	0.210
C3-4	20	7062.00	*8697.00	*8697.00	8697.00	8697.00	0.000	0.000	0.720
C3-5	25	10198.00	12480.70	12539.00	12506.00	12506.00	0.203	0.000	3.680
C3-6	30	11043.00	13428.10	13506.00	13463.00	13468.00	0.297	0.037	7.530
C3-7	35	10859.00	12223.60	12365.00	12265.00	12274.00	0.412	0.073	8.790
C3-8	40	13736.00	15201.00	15427.00	15226.00	15285.00	0.553	0.387	24.360
C3-9	45	14420.00	16042.50	16437.00	16147.00	16183.00	0.876	0.223	47.840
C3-10	50	18903.00	20963.10	21284.00	21012.00	21076.00	0.539	0.305	123.280
Avg.		6539.03	8126.96	8284.20	8166.33	8182.07	0.620	0.151	14.488

5 Conclusion

This paper has introduced a quick and effective tabu search heuristic for a sequence-dependent and time-dependent scheduling problem on a single machine. The heuristic relies on classical shift and swap moves, but it uses a surrogate objective function to quickly identify a set of promising solutions. In addition, two diversification mechanisms are applied to ensure a broad exploration of the search space. Comparing the computational results of the heuristic with those of a recent branch-and-cut algorithm indicates that the heuristic can find high quality solutions in very short computing times. When both algorithms can solve an instance to optimality, the tabu search heuristic is faster, while for instances that cannot be solved optimally by the branch-and-cut, the heuristic produces a better solution in less time.

Table 8 Results for the fourth set of instances

Inst	n	FIXED	LB	UB	BKS	$z(s)$	$\%_{LB}$	$\%_{BKS}$	CPU
D1-1	5	4180.00	*5515.00	*5515.00	5515.00	5515.00	0.000	0.000	0.000
D1-2	10	9571.00	*10848.00	*10848.00	10848.00	10848.00	0.000	0.000	0.080
D1-3	15	10650.00	*13632.00	*13632.00	13632.00	13632.00	0.000	0.000	0.470
D1-4	20	4358.00	6112.32	6346.00	6346.00	6346.00	3.823	0.000	1.290
D1-5	25	5897.00	8024.35	8526.00	8363.00	8386.00	4.507	0.275	4.670
D1-6	30	7932.00	10130.30	11003.00	10643.00	10741.00	6.028	0.921	12.610
D1-7	35	9364.00	12122.70	13220.00	12609.00	12663.00	4.457	0.428	27.360
D1-8	40	11538.00	13842.00	15725.00	14833.00	14996.00	8.337	1.099	73.120
D1-9	45	10609.00	14441.00	16282.00	15022.00	15195.00	5.221	1.152	99.810
D1-10	50	10964.00	15043.70	17335.00	15933.00	16074.00	6.849	0.885	172.730
D2-1	5	2838.00	*5080.00	*5080.00	5080.00	5080.00	0.000	0.000	0.000
D2-2	10	3807.00	*6467.00	*6467.00	6467.00	6467.00	0.000	0.000	0.080
D2-3	15	7833.00	*9797.00	*9797.00	9797.00	9797.00	0.000	0.000	0.330
D2-4	20	8373.00	11035.80	11242.00	11117.00	11117.00	0.736	0.000	0.900
D2-5	25	12470.00	14357.90	14694.00	14577.00	14577.00	1.526	0.000	2.710
D2-6	30	14867.00	17408.00	17906.00	17619.00	17619.00	1.212	0.000	6.280
D2-7	35	19894.00	22625.40	23310.00	22884.00	22895.00	1.192	0.048	13.560
D2-8	40	17392.00	21013.00	22213.00	21294.00	21370.00	1.699	0.357	35.950
D2-9	45	20220.00	24297.70	25653.00	24679.00	24780.00	1.985	0.409	68.120
D2-10	50	23943.00	26918.30	28537.00	27228.00	27228.00	1.151	0.000	106.610
D3-1	5	3777.00	*5888.00	*5888.00	5888.00	5888.00	0.000	0.000	0.000
D3-2	10	6797.00	*8616.00	*8616.00	8616.00	8616.00	0.000	0.000	0.080
D3-3	15	11451.00	*13661.00	*13661.00	13661.00	13661.00	0.000	0.000	0.320
D3-4	20	13670.00	16830.80	16988.00	16988.00	16988.00	0.934	0.000	1.670
D3-5	25	18689.00	21631.10	21867.00	21795.00	21847.00	0.998	0.239	3.330
D3-6	30	22062.00	25332.00	25835.00	25562.00	25637.00	1.204	0.293	11.440
D3-7	35	26502.00	29336.50	30064.00	29687.00	29744.00	1.389	0.192	22.930
D3-8	40	33723.00	36806.20	37759.00	37030.00	37050.00	0.662	0.054	46.480
D3-9	45	31865.00	34645.20	35978.00	35085.00	35239.00	1.714	0.439	99.580
D3-10	50	35868.00	39878.90	41000.00	40203.00	40377.00	1.249	0.433	168.680
Avg.		14036.80	16711.24	17366.23	16966.70	17012.43	1.896	0.241	32.706

Acknowledgements This research was partially funded by the Natural Sciences and Engineering Research Council of Canada under grant 227837-04. This support is gratefully acknowledged. We are also grateful to three anonymous referees for their valuable comments.

References

Albiach, J., Sanchis, J. M., & Soler, D. (2007). An asymmetric tsp with time windows and with time-dependent travel times and costs: an exact solution through a graph transformation. *European Journal of Operational Research*. doi:10.1016/j.ejor.2006.09.099.

Alidaee, B., & Womer, N. K. (1999). Scheduling with time dependent processing times: review and extensions. *Journal of the Operational Research Society*, *50*, 711–720.

Barketau, M. S., Cheng, T. C. E., & Ng, C. T. (2005). *Batch scheduling of step deteriorating items*. Working paper, Department of Logistics, The Hong Kong Polytechnic University.

Bigras, L. P., Gamache, M., & Savard, G. (2006). *The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times*. Les Cahiers du GERAD, G-2006-15.

Cheng, T. C. E., & Ding, Q. (2001). Single machine scheduling with step-deteriorating processing times. *European Journal of Operational Research*, *134*, 623–630.

Cheng, T. C. E., Ding, Q., & Lin, B. M. T. (2004). A concise survey of scheduling with time dependent processing times. *European Journal of Operational Research*, *152*, 1–13.

Donati, A. V., Montemanni, R., Casagrande, N., Rizzoli, A. E., & Gambardella, L.M. (2006). Time dependent vehicle routing problem with a multi ant colony system. *European Journal of Operational Research*. doi:10.1016/j.ejor.2006.06.047.

Gendreau, M. (2003). An introduction to tabu search. In F. Glover & G. A. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 37–54). Boston: Kluwer Academic.

Glover, F., & Laguna, M. (1997). *Tabu search*. Boston: Kluwer Academic.

Haghani, A., & Jung, S. (2005). A dynamic vehicle routing problem with time-dependent travel times. *Computers and Operations Research*, *32*, 2959–2986.

Ichoua, S., Gendreau, M., & Potvin, J.-Y. (2003). Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, *144*, 379–396.

Jeng, A. A. K., & Lin, B. M. T. (2004). Makespan minimization in single-machine scheduling with step-deterioration of processing times. *Journal of the Operational Research Society*, *55*, 247–256.

Khammuang, K., Abdekhodaee, A., & Wirth, A. (2007). On-line scheduling with forbidden zones. *Journal of the Operational Research Society*, *58*, 80–90.

Lahlou, C., & Dauzère-Pérès, S. (2006). Single-machine scheduling with time window-dependent processing times. *Journal of the Operational Research Society*, *57*, 133–139.

Leung, J. Y. T., Ng, C. T., & Cheng, T. C. E. (2006). Minimizing sum of completion times for batch scheduling of jobs with deteriorating processing times. *European Journal of Operational Research*. doi:10.1016/j.ejor.2006.03.067.

Malandraki, C., & Daskin, M. S. (1992). Time dependent vehicle routing problems: formulation, properties and heuristic algorithms. *Transportation Science*, *26*, 185–200.

Malandraki, C., & Dial, R. B. (1996). A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research*, *90*, 45–55.

Mosheiov, G. (1995). Scheduling jobs with step-deterioration: minimizing makespan on a single and multi-machine. *Computers and Industrial Engineering*, *28*, 869–879.

Picard, J. C., & Queyranne, M. (1978). The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research*, *26*, 86–110.

Stecco, G., Cordeau, J.-F., & Moretti, E. (2008). A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times. *Computers and Operations Research*, *35*, 2635–2655.

Taillard, É. D. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, *23*, 661–673.