J Sched (2015) 18:89–105
DOI 10.1007/s10951-014-0395-8

Metaheuristics for a scheduling problem with rejection and tardiness penalties

Simon Thevenin · Nicolas Zufferey · Marino Widmer

Received: 10 December 2012 / Accepted: 20 August 2014 / Published online: 5 September 2014

Abstract In this paper, we consider a single-machine scheduling problem (P) inspired from manufacturing instances. A release date, a deadline, and a regular (i.e., non-decreasing) cost function are associated with each job. The problem takes into account sequence-dependent setup times and setup costs between jobs of different families. Moreover, the company has the possibility to reject some jobs/orders, in which case a penalty (abandon cost) is incurred. Therefore, the problem at hand can be viewed as an order acceptance and scheduling problem. Order acceptance problems have gained interest among the research community over the last decades, particularly in a make-to-order environment. We propose and compare a constructive heuristic, local search methods, and population-based algorithms. Tests are performed on realistic instances and show that the developed metaheuristics significantly outperform the currently available resolution methods for the same problem.

Keywords Scheduling · Metaheuristics · Order acceptance · Setups

S. Thevenin · N. Zufferey (✉)
Geneva School of Economics and Management (GSEM), University of Geneva, Geneva, Switzerland
e-mail: n.zufferey@unige.ch

S. Thevenin
e-mail: simon.thevenin@unige.ch

M. Widmer
DIUF Decision Support & Operations Research, University of Fribourg, Fribourg, Switzerland
e-mail: marino.widmer@unifr.ch

1 Introduction

In this paper, we consider the problem of scheduling n jobs on a single machine in order to minimize regular (i.e., non-decreasing) objective functions ($f_j(C_j)$, where C_j is the completion time of job j). We also take into account sequence-dependent setup costs (c_{ij}) and sequence-dependent setup times (s_{ij}) between any successively performed jobs i and j of different families. In addition, there is the possibility to reject some jobs, they are then said to be unperformed and a penalty cost (u_j) must be paid. For each job j, a release date (r_j), a deadline ($\bar{d}_j$), and a processing time (p_j) are given. Using the three-field notation introduced in Graham et al. (1979), the problem can be denoted as $(1 \mid r_j, s_{i,j}, \bar{d}_j \mid F_l(\sum f_j(C_j), \sum u_j, \sum c_{i,j}))$. The objective is not necessarily to minimize the number of rejected jobs, but rather to minimize a linear combination of the three above-mentioned components. We design a constructive heuristic, local search algorithms, and population-based methods for (P). Baptiste and Pape (2005) developed a branch and bound algorithm in a constraint programming framework to solve (P). The authors proposed a lower bound and a dominance rule. Their method was able to solve instances with up to 30 jobs, even if some instances with 24 jobs are still open. To our knowledge, it is the only paper addressing such a problem.

In the literature, the two most popular regular objective functions are the sum of completion times ($\sum C_j$) and the sum of tardiness ($\sum T_j$), with their weighted versions ($\sum w_j \cdot T_j$ and $\sum w_j \cdot C_j$). Du and Leung (1990) showed that the single-machine scheduling problem which aims to minimize the sum of weighted tardiness ($1||\sum w_j \cdot T_j$) is NP-hard. As it is a particular case of (P), the latter is NP-hard too. The one machine scheduling problem which aims to minimize the sum of completion times with release

dates ($1|r_j|\sum C_j$) is strongly NP-hard (Pinedo 2008), unless the release dates are all equal, whereas the same problem without release dates ($1||\sum C_j$) and its weighted version ($1||\sum w_j \cdot C_j$) are solvable in polynomial time.

This paper is a significant extension of Thevenin et al. (2012), where a greedy algorithm and a tabu search are proposed for (P). A key feature of the proposed tabu search is the joint use of different moves. Metaheuristics using several neighborhoods are common, we can for instance mention the variable neighborhood search and oscillation strategy. In most cases, neighborhoods are sequentially used either in a random fashion or a strategic one. Using jointly multiple neighborhoods in tabu search was proven to be successful for the maximum weighted clique problem in Wu et al. (2012). In Lü et al. (2009), the authors present a comparison of different combinations of moves for the unconstrained binary quadratic problem. The contributions of this paper are the following. Several fields of application for (P) are identified; a mathematic formulation is given and tested on small size instances. A more efficient tabu search is proposed, which includes a diversification strategy. We investigate the use of hybrid metaheuristics for (P) (namely two adaptive memory algorithms). We also propose a set of difficult instances which allows to conduct a better comparative study between these algorithms.

The paper is organized as follows. A formal description of (P) is given in Sect. 2 (including a mathematical formulation), as well as possible practical applications of (P). As the literature review is significant, it is presented in a dedicated Sect. 3. The constructive and local search methods for (P) are proposed in Sect. 4, whereas the population-based methods are designed in Sect. 5. Results and comparisons can be found in Sect. 6. A conclusion ends up the paper, along with possible extensions.

2 Description and motivation of problem (P)

We are interested in a one machine scheduling problem with n jobs. For each job j are given the following data:

- p_j: the *processing time* of job j.
- r_j: the *release date* of job j. It is the time from which it is possible to start processing job j. In manufacturing systems, it could be the time at which the raw material is expected to be delivered to the production system.
- d_j: the *due date* of job j. It represents the time after which the satisfaction of the customer decreases (this is represented by a non-decreasing cost function $f_j(C_j)$). More formally, d_j can be defined as the date from which $f_j(.)$ starts to be larger than zero.
- $\bar{d}_j$: the *deadline* of job j. Scheduling a job j after its deadline is not possible. It could for instance be the time after which the penalty cost of scheduling late is higher than the abandon cost, or the time after which the customer does not want to be served. Thus, each job j must be performed within a time window $[r_j, \bar{d}_j]$.
- u_j: the *abandon cost* of job j, which is the penalty encountered if job j is unperformed.

Jobs belong to different *families*, which correspond to different types of product. When the machine successively processes two jobs i and j of different families, a *setup* must be performed. This implies a setup time s_{ij}, which is the time to tune the machine, and a setup cost c_{ij} (to pay employees which setup the machine and the needed material). At the beginning, the machine is in an initial state, which we represent by a dummy job 0 such that $p_0 = 0$. s_{0j} (resp. c_{0j}) is the requested setup time (resp. cost) between the initial state and job j, which must be taken into account if j is scheduled first. The objective function $\sum_j f_j(C_j)$ is a sum, over all the jobs, of regular (i.e., non-decreasing) functions depending on the completion times C_j. We consider general cost functions, allowing our algorithms to tackle different problems, or even to use different cost functions for the different jobs.

A mathematical formulation of (P) is now presented. The formulation is linear if for all job j, the cost function $f_j(C_j)$ is linear. We assume that $x_{jk} = 1$ if job k follows job j, 0 otherwise; $z_j = 1$ if j is unperformed, 0 otherwise; and t_j is the starting time of job j. For the need of the formulation, we artificially add a last job $n+1$, with $p_{n+1} = 0$, $s_{(j)(n+1)} = 0$, $c_{(j)(n+1)} = 0, \forall\, j$.

$$\min \left[\sum_{j=0}^{n} \sum_{k=1}^{n+1} x_{jk}\, c_{jk} \sum_{j=1}^{n} \left[f_j(C_j) + z_j\,(u_j - f_j(r_j)) \right] \right] \tag{1}$$

s.t.

$$C_j = t_j + p_j \qquad \forall j \tag{2}$$

$$t_j \geq C_k + s_{kj}\, x_{kj} + (x_{kj} - 1)\, \bar{d}_k \quad \forall j, k \tag{3}$$

$$t_j \geq r_j \text{ and } C_j \leq \bar{d}_j + z_j\,(r_j - \bar{d}_j) \quad \forall j \tag{4}$$

$$z_j + \sum_{k=1}^{n+1} x_{jk} = 1 \qquad \forall j \neq n+1 \tag{5}$$

$$z_j + \sum_{k=0}^{n} x_{kj} = 1 \qquad \forall j \neq 0 \tag{6}$$

$$\sum_{j=0}^{n+1} x_{j0} = 0 \text{ and } \sum_{j=0}^{n+1} x_{(n+1)j} = 0 \text{ and } t_0 = 0 \tag{7}$$

$$x_{jk} \in \{0,1\} \qquad z_j \in \{0,1\} \qquad \forall j \tag{8}$$

Equation (1) gives the objective function: it is the sum of setup costs, and for each job, its cost is $f_j(C_j)$ if it is performed, and its unperformed penalty is u_j otherwise. The

completion time of each unperformed job is virtually set to its release date r_j by Eq. (4). For this reason in Eq. (1), for each unperformed job, we add its unperformed cost u_j and remove the cost $f_j(r_j)$ associated with its release date. Equation (2) gives the completion time. Equation (3) is used to compute the starting time of each job. If j follows k, j must start after the end of k plus the required setup time, otherwise $(x_{kj} - 1)$ is equal to -1, and the resulting constraint is less restrictive than $C_j \geq 0$, as $\bar{d}_k \geq C_k$. Equation (4) enforces each performed job to be scheduled within its time window and each unperformed job to be scheduled at r_j. Equations (5) and (6) specify that each job must have a successor and a predecessor (or it is unperformed). Equation (7) constraints job 0 to have no predecessor, job $n+1$ to have no successor, and the starting time of the schedule is 0.

Three possible realistic situations associated with (P) are described below. First, assume the situation where a company which produces packaging parts (plastic bottles, cans,…) wants to schedule its production. Bottles are made by injection moulding machines, different types of machine are available in the factory, but each job is dedicated to a single machine. As a consequence, the problem can be treated as a series of one machine scheduling problems. Depending on the products being processed consecutively, different operations have to be performed: cleaning, changing the plastic colors and type, changing the mould,…Those operations are time and cost expensive, and have to be minimized. Obviously, processing consecutively different orders of the same package (for the same product) do not incur any setups. Moreover, an order cannot be treated before the delivery of the associated raw materials, and its duration depends on the number of units to produce. As products are specific for each client, production is run only after that the order has been given. If the production capacity is overloaded, the company will deliver late. This causes customers dissatisfaction, which can be modeled by a non-decreasing cost function depending on the completion time of the job. Instead of being very late, the company can subcontract an order [note that if only a part of the jobs can be subcontracted, others can be associated with infinite rejection penalties in (P)]. Examples of scheduling problem in injection moulding plants can be found in Nagarur et al. (1997) and Goslawski et al. (2014).

Another possible application occurs in textile dyeing companies. To be colored, textiles are put in dyebath, the duration of the stay mainly depends on the used coloring method. Between different jobs, the dyebath must be cleaned. The cleaning duration depends on the dyes being used consecutively. If the same solution is used, no cleaning operation is needed. If similar colors are processed consecutively, a fast cleaning occurs, whereas in case of totaly different colors, an heavy cleaning is needed, meaning more time and cleaning products. Before to start an order, the company has to wait for the necessary raw material to be delivered (dies, fabrics, mordants,…). We consider the case of a small company having only one dyebath. If the production capacity is overloaded, orders are delivered late. In the worst case, they are canceled and customers have to wait for the next week to be delivered, which cause dissatisfaction modeled by a rejection cost depending on the importance of the customer. Once an order has been rejected, it is considered again in the next week planning, but its rejection penalty increases. For more information on scheduling problems in textile dyeing manufacture, readers are referred to the recent paper of (Hsu et al. 2009).

The last field of application concerns agile satellite scheduling. Customers ask for a set of images of the Earth. Images are made of several pixels, each one corresponding to a snapshot. The required time to get a total picture is related to the size of the area to be imaged. Customers give a time window corresponding to the date at which the satellite is at a good position to take the picture. However, it is possible to take the picture after the time window, in which case the angle is not optimal leading to dissatisfaction of the customer. We can assume that the release date is defined as the earliest possible time to take the picture. Thus, starting before the release date is not allowed. To summarize, a possible time window $[r_j, \bar{d}_j]$ is given, but between d_j and $\bar{d}_j$ the quality of the picture is decreasing. As launching a satellite is expensive, it is often funded by several companies. As a consequence, satellites are often overloaded and it is often necessary to postpone several requests. If it is not possible to take the picture, a negotiation occurs to schedule the image later, this incurs a cost of negotiation (phone calls, salary of persons making negotiations,…) and frustration of customers. This problem also implies setup times for moving the satellite from one zone of interest to the next, and setup costs correspond to the energy used during the setup operations (indeed the energy is a limited resource on satellites). Examples of satellite scheduling problem can be found in Harrison et al. (1999), Lemaitre et al. (2002), Wei-Cheng and Chang (2005), and Zufferey et al. (2008).

3 Literature review

In this section, we present the methods developed for some problems related to (P). A specific attention is given to dispatching rules, local search algorithms, order acceptance problems (OAP), and evolutionary methods. The readers are referred to Baptiste and Pape (2005) for an insight of exact approaches proposed for problems related to (P), and to Pinedo (2008) for an overview of scheduling problems.

For scheduling problems, we often encounter *dispatching rules*, which allow to define a priority order for the jobs. In some particular cases, dispatching rules give an optimal sequence of jobs, whereas in other cases they are greedy

heuristics able to quickly obtain relatively good solutions. An efficient dispatching rule for the one machine scheduling problem which aims to minimize the weighted tardiness ($1||\sum w_j \cdot T_j$) is the ATC (apparent tardiness cost) (Vepsalainen and Morton 1987). In ATC, at any time t where the machine is available, the next job j to be scheduled is the one which maximizes $\frac{w_j}{p_j}\, exp[-\frac{max(d_j-p_j-t,0)}{k\cdot\bar{p}}]$, where $\bar{p}$ is the average processing time, and k is a parameter. Lee et al. (1997) extend the rule for the case with sequence-dependent setup times, and Shin et al. (2002) make an adaptation of the rule where the objective function is the maximum lateness ($1|r_j, s_{ij}|L_{max}$), which is denoted MATCS (modified apparent tardiness cost with setups). A MATCS heuristic, is proposed for a problem related to (P) as well as a dynamic dispatching rule. Yang and Geunes (2007) were interested in the single-machine scheduling problem, with the possibility of not performing some jobs. A global deadline is given and the jobs cannot be scheduled after it. Moreover, the processing time of each job can be reduced by a compression operation. The objective function to maximize is the profit of each performed job, minus the compression and tardiness costs. The authors propose a GRASP (greedy randomized adaptive search procedure) algorithm where a schedule is built by a randomized dispatching rule and an approximation algorithm obtained by the adaptation of an algorithm for the intervals selection problem. Akturk and Ozdemir (2001) tackle the single-machine problem with release dates to minimize the weighted tardiness ($1|r_j|\sum_j w_j \cdot T_j$). They propose a dominance rule which provides a sufficient condition for local optimality. The rule is implemented to improve two dispatching rules, a GRASP and a local search algorithm. Note that for (P), the use of overall regular cost functions makes it difficult to propose efficient dispatching rules. We rather propose a constructive heuristic which fits better with the use of overall cost functions.

Local search methods are often efficient for scheduling problems. The most used neighborhood structures for single-machine scheduling problem are *Swap* and *Reinsert* (Shin et al. 2002; Laguna et al. 1991). *Reinsert* consists in taking a job in the schedule and moving it to another position. *Swap* consists in swapping two jobs in the schedule. The authors conclude that *Reinsert* is better than *Swap*, but that using a *Hybrid* neighborhood yields better results (*Hybrid* is simply the union of *Swap* and *Reinsert*). Jouglet et al. (2008) consider the one machine scheduling problem with release dates which consists in minimizing the weighted tardiness ($1|r_j|\sum w_j \cdot T_j$). They show that the use of dominance rules allows to improve tabu search for scheduling problems. Among other metaheuristics for problems related to (P), we can mention (Bożejko 2010), where a scatter search is presented to solve the one machine scheduling problem with setup times to minimize the weighted tardiness ($1|s_{ij}|\sum w_j \cdot T_j$), and Kirlik and Oğuz (2012) propose a variable neighborhood descent for the same problem. The main differences between (P) and the above cited problems are that (P) includes release dates, deadlines, and the possibility of rejecting some jobs, which considerably changes the way of tackling the problem. Having possible time windows imposes to define a way of maintaining feasibility. Moreover, different types of moves must be defined to take into account rejections. Local search methods have also been proposed for different scheduling environments. For instance, Anghinolfi and Paolucci (2007) propose a local search method using features of tabu search, simulated annealing, and variable neighborhood search for a parallel machines scheduling problem where the objective is the weighed tardiness.

As explained in Slotnick (2011), the problem of minimizing the penalties incurred by rejected jobs, as it is also defined in (P), is equivalent to the OAP. In OAP, a company has to decide which orders to accept in the aim of maximizing its revenue, which is the sum of the gains associated with each performed job minus some costs related to the sequence in which jobs are performed. OAP has been studied in a wide range of scheduling environments during the last decades, and it is particularly relevant in a make-to-order production system (Zorzini et al. 2008). A review is presented in Slotnick (2011). Bilgintürk Yalçın et al. (2007) and Oğuz et al. (2010) tackled an OAP with one machine, release dates, deadlines, and sequence-dependent setup times. Their objective is to maximize the sum of the gains associated with each performed job minus a weighted tardiness penalty. This problem differs from (P) by the fact that setup costs are not taken into account, there are no job families, and in (P) we consider general cost functions. The authors propose a MILP (mixed integer linear programming) formulation for the problem, which is only able to solve instances with up to 15 jobs, as well as constructive and local search heuristics. The local search works in two steps: select the orders first, then find a good sequence. The same problem is studied in Cesaret et al. (2012), where the authors show that making simultaneously sequencing and order accepting decisions improves the results. Their approach consists in a tabu search with *Swap* moves. Note that in their version of *Swap*, it is allowed to exchange a performed job with a rejected one. Other relevant papers on OAP include Nobibon and Leus (2011) and Nobibon et al. (2009), where some orders can be rejected whereas others must be performed, ignoring release dates and setups, which considerably change the nature of the problem. Zhang et al. (2009) propose a dynamic programming algorithm and an approximation algorithm for the single-machine scheduling problem with rejection and release date. The objective function is the sum of rejection penalties plus the makespan. Shabtay et al. (2012) consider the order acceptance and scheduling problem in a single-machine environ-

ment. Each job is available at time 0 and there is no deadline. By nature, such problems involve two objectives: F_1 is related to the completion times, and F_2 is the sum of rejection penalties. If the global objective is the sum of F_1 and F_2, under some conditions, they show that the problem can be solved in polynomial time for different objectives F_1. However, the authors note that the problem of minimizing F_1 (resp. F_2) subject to $F_2 \leq K$ (resp. $F_1 \leq R$) is NP-hard (where K and R are given bounds). Also, finding the Pareto set corresponding to these two objectives is NP-hard. For these three last versions, the authors propose approximation methods and exact methods for some special cases. The OAP has also been studied in different scheduling environments. For instance, Xiao et al. (2012) studied the OAP in a flow shop environment. The objective is to maximize the gains of accepted jobs minus the weighted tardiness penalties. The authors propose a simulated annealing which optimizes sequentially the set of jobs to accept and the sequence of jobs.

Evolutionary metaheuristics (e.g., genetic algorithms, adaptive memory algorithms) were successfully applied to scheduling problems. Among applications of genetic algorithms to problems related to (P), we can mention (Ribeiro et al. 2010; Rom and Slotnick 2009; Sels and Vanhoucke 2011). To be competitive, all these algorithms use a local search to intensify the search. Ribeiro et al. (2010) solve a single-machine scheduling problem minimizing the weighted earliness and tardiness penalties, with due time windows and setup times $(1|s_{ij}|h_j \cdot E_j + w_j \cdot T_j)$. The solutions are initialized by a GRASP using five different dispatching rules. In order to improve the robustness of the algorithm, five different crossovers are proposed. At each iteration, a crossover operator is chosen randomly, giving higher probabilities to the ones which produced good solutions in the past generations. Best solutions are improved by a local search method at each generation. Path relinking is used every five generations. Rom and Slotnick (2009) propose a genetic algorithm for the OAP with tardiness costs, and each solution is represented as a sequence containing all jobs. The recombination operator is a double-points crossover, and some jobs are rejected within the evaluation phase during which the schedule is greedily built from the sequence. Diversification is maintained by four ways: checking for duplicate solutions (having the same costs), using mutation, varying the population size, using two separate populations with migration between them. A local search is also used to improve solutions. Sels and Vanhoucke (2011) tackle a single-machine scheduling problem with release dates, while minimizing the maximum lateness $(1|r_j|L_{max})$, with a genetic algorithm. The authors tested several settings and obtained the best results with tournament selection, a position-based crossover (i.e., select randomly a certain number of positions in the sequence vector of the initially empty offspring, then fill these positions with jobs copied from the first parent, finally the jobs of the second parent which are not present in the offspring solution keep their order and fill the remaining empty positions), a single swap mutation, and a local search which reinserts the jobs having the highest costs. Kellegöz et al. (2008) present a comparison between different crossover operators for $(1||\sum w_j \cdot T_j)$, and the best results are obtained by crossovers preserving the position or order of the jobs [this will also be the case for (P)]. In Zufferey et al. (2008), the authors propose a tabu search algorithm and an adaptive memory algorithm (AMA) for a satellite range scheduling problem with time windows, where the number of unperformed jobs has to be minimized. They take advantage of the graph coloring literature to tackle the problem, and show that AMA performs slightly better than tabu search. Note by the way that several unified views of evolutionary metaheuristics have been proposed in the literature (Bo et al. 2011; Hertz and Kobler 2000; Taillard et al. 2001). AMA is deeply discussed in Taillard et al. (2001). The main difference between AMA and genetic algorithms with local search is that AMA uses a simplified evolution process, but a more important role is given to local search. Evolutionary methods have proven to be successful in many domains including scheduling problems. We propose in this paper to test the efficiency of such methods for (P), and show how they can be coupled with a tabu search algorithm.

4 Constructive and local search methods for (P)

In this section, we present a greedy algorithm, a descent method as well as a tabu search for (P). The latter metaheuristic contains several powerful ingredients: the use of four neighborhoods at each iteration, a restriction technique which accelerates the algorithm by ignoring non-promising moves, a diversification mechanism, incremental computation, and a tabu status based on solution values (and not only on solution attributes).

In our model, a solution s of (P) can be represented by a pair $\hat{s} = (\Pi, \Omega)$, where Π gives the sequence of the performed jobs (but not the starting time of each job), and Ω is the set of the unperformed jobs. Let $\pi(\kappa)$ be the index of job scheduled at position κ. Thus, a sequence Π with n' elements can be denoted by $\Pi = [\pi(1), \pi(2), \ldots, \pi(n')]$ (where $n' \leq n$). Assuming $\pi(0) = 0$, the cost function can therefore be computed by

$$\sum_{j \in \Omega} u_j + \sum_{\kappa=1}^{n'} f_{\pi(\kappa)}(C_{\pi(\kappa)}) + \sum_{\kappa=0}^{n'-1} c_{\pi(\kappa)\pi(\kappa+1)} \tag{9}$$

Since the objective functions are regular, we can easily build the schedule from a given list of jobs by setting the starting time of a job to the earliest feasible time as follows:

$$t_{\pi(j)} = \max\{C_{\pi(j-1)} + s_{\pi(j-1)\pi(j)},\ r_{\pi(j)}\} \tag{10}$$

A *timing algorithm* takes as input a solution representation $\hat{s} = (\Pi, \Omega)$. Its output is a solution s which indicates the value C_j of each performed job j. It builds the schedule from the first job to the last with Eq. (10), and jobs are rejected only if they are shifted after their deadlines.

A combinatorial optimization problem is defined by a set of feasible solutions S and an objective function f. A solution is called feasible if it satisfies all the constraints. The goal is to minimize (or maximize) f over S. For some problems, there exists no algorithm able to find an optimal solution in a polynomial time. In contrast, heuristics are able to find satisfying solutions in a reasonable amount of time. Metaheuristics are higher level approaches, which can combine problem-specific methods. There are three main kinds of (meta)heuristics: constructive heuristics (e.g., the greedy algorithm), local search methods (e.g., tabu search, simulated annealing), and evolutionary algorithms (e.g., genetic algorithms, adaptive memory algorithms, ant colonies). The reader is referred to Gendreau and Potvin (2010) and Zufferey (2012) to have an overview on metaheuristics and principles to efficiently adapt such methods.

4.1 Greedy algorithm for (P)

A *constructive* heuristic builds a solution by starting from scratch. It adds elements step by step to the solution until it becomes a complete solution. A *greedy* heuristic for (P) is introduced in this subsection. It will then be used to generate initial solutions for tabu search, which is a *local search* method (Sect. 4.2). The latter will then be used as an intensification operator within some *evolutionary* algorithms (Sect. 5).

The greedy algorithm for (P) begins with an empty schedule. Jobs are then taken one by one and placed in the schedule. Each step consists in selecting a non-considered job and inserting it within the schedule at minimum cost (there is also the possibility of letting the job unperformed). The position which minimizes the cost is chosen (ties are broken randomly). As the jobs are scheduled as early as possible with respect to the sequence, inserting a job before another can shift the entire schedule, thus it is necessary to be careful when computing the cost function at each step. By shifting a part of the schedule to the right, some jobs may end after their deadlines: they will then enter in the unperformed set. Note that the cost function is computed after having shifted and removed the jobs, thus jobs become unperformed only if it is better to do it. As we have to compute the cost function for each job and each possible position, the cost function is computed approximately n^2 times. Computing the cost function can be done in $O(n)$ in the worst case. Thus the greedy algorithm runs in $O(n^3)$. An *incremental* cost function is used to compute the cost of inserting the jobs, which allows to speed up the algorithm. The order in which jobs are inserted into the schedule has an influence on the results. After having tested several possibilities to sort the jobs, we decided to order the jobs by increasing *slack times* ($\bar{d}_j - r_j - p_j$). Ties are broken by decreasing u_j's, and if there remain ties, they are broken randomly.

4.2 Local search approaches for (P)

Local search methods need an initial solution as input, and then explore the solution space by going from the current solution to a neighbor solution. A *neighbor* solution is often obtained by making a slight modification on the current solution, called a *move*. The *neighborhood* $N(s)$ of a solution s is the set of solutions obtained by applying to s all possible moves. From a current solution s, the descent algorithm selects its neighbor solution s' as the best solution in $N(s)$. The main issue with this method is that it is likely to bring the search in a local optimum. To overcome this issue, *tabu search* makes use of recent memory, with a so called tabu list. It forbids to perform the reverse of the moves done during the last tab (parameter) iterations, where tab is called *tabu tenure*.

In order to adapt tabu search for (P), we propose to use the four following neighborhood structures.

- *Reinsert* (and shift and drop) moves a job from a position κ to another position κ'. Once the move is performed, jobs are shifted to be scheduled as early as possible. Jobs can be dropped if they end after their deadlines.
- *Swap* (and shift and drop) swaps two jobs. Once the move is performed, the schedule is shifted and jobs can be dropped if they end after their deadlines. If *Swap* is jointly used with *Reinsert*, swapping two consecutive jobs is forbidden, because adjacent pairwise interchange is already included in *Reinsert*.
- *Add* (and shift and drop) adds a job from the unscheduled set at a position κ, and then the schedule is shifted. Jobs can be dropped if they end after their deadlines.
- *Drop* (and shift) drops a job, and then shifts the right part of the schedule to the left. Because of the shifting process, dropping a job can reduce the value of the cost function.

While performing these moves, the cost associated with some of the jobs will not change and *incremental* computation can be used. None of these neighborhoods can be used alone, as a single neighborhood structure does not allow to reach all the solutions of the solution space (which means that the search space would not be *connected*). Several papers (e.g., Shin et al. 2002; Laguna et al. 1991) confirmed that using the *union* of several types of moves leads to better results. Therefore, at each iteration, the selected neighbor solution will be generated from the current solution by performing the

best move among the four above proposed types of moves (restrictions will occur for tabu search).

On the one hand, we develop a descent algorithm starting from an initial solution generated by the greedy algorithm. On the other hand, we propose a tabu search by adding tabu structures to the descent algorithm. We design four tabu structures. (1) When a job has been added into the schedule, it cannot be dropped during τ_1 iterations, and it cannot be moved (by any of the moves) during τ_2 iterations. (2) When a job has been dropped, it cannot be added during τ_3 iterations. Note that, for this tabu structure, the job is considered dropped only by the above fourth neighborhood (i.e., *Drop*) and not by the shift and drop procedure. (3) When a job has been reinserted, it cannot be moved during τ_2 iterations, and it cannot return between its two previous adjacent jobs during τ_4 iterations. (4) When a job has been swapped, it cannot be moved during τ_2 iterations, and it cannot return between its two previous adjacent jobs during τ_4 iterations. $\tau_1, \tau_2, \tau_3, \tau_4$ are parameters of the algorithm. We propose to have $\tau_2 < \tau_4$ and $\tau_2 < \tau_1$ because the tabu structure associated with τ_2 is more restrictive than the one associated with τ_4 and τ_1, while τ_3 does not depend on other parameters. Note that the same parameter τ_2 is used in the tabu status (1), (3) and (4), as it corresponds to a similar restriction, which forbids to move again the job. A similar remark holds for τ_4, which is used for the tabu status (3) and (4).

We propose now some mechanisms and procedures to improve the above proposed tabu search approach. First, as it is time consuming to evaluate all the neighbor solutions at each iteration, we propose to only evaluate a promising sample of move. Then, a diversification mechanism and an additional tabu status are developed.

Restrictions on the neighborhood. On the one hand, we do not schedule a job j at position κ if the job currently scheduled at this position ends before the release date of j (i.e., $r_j > C_{\pi(\kappa)}$). If such a move is performed, j is going to be shifted to its release date, bringing with it all jobs scheduled between position κ and time r_j. This leads to useless idle time, and thus higher cost. As a consequence, such poor moves will be ignored. On the other hand, to reduce the computational effort needed for a single iteration, each move is only evaluated with a probability of 25 %. Such a neighborhood reduction technique is commonly used to improve tabu search approaches, as it usually increases the diversification skill of the algorithm without much decreasing its intensification ability.

Diversification mechanism. We make use of a long-term memory in a diversification procedure consisting in dropping the *oldest* jobs of the schedule. We define the *age* of a job as the number of iterations since when it is in the schedule (and not in the unperformed set). The diversification procedure drops the 30 % oldest jobs out of the schedule. It is applied every 500 iterations of tabu search. When a job is dropped, it gets the corresponding tabu status.

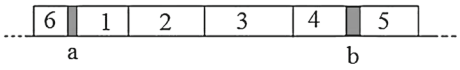

Fig. 1 Example of equivalent solution

Additional tabu status. Preliminary experiments showed that there exist many solutions in the neighborhood with the same value. In the example depicted in Fig. 1, we assume that jobs 1, 2, 3, and 4 belong to the same family, thus there is no setup time and cost between them. We also assume that these jobs have the same release date a, and that the cost function of all jobs (f_j) is constant over $[a, b]$. Jobs 1 to 4 can be positioned in any order before b without changing the cost, as the beginning of job 5 will remain the same, and the remainder of the schedule will be unchanged. Note that this situation is specific to the case where the f_j's are constant over some intervals. This has the effect to bring the search into a plateau from which it is hard to escape. Once the solution cannot be improved, only moves leading to an equivalent solution are performed (i.e., moves with the same objective function value). To tackle this issue, Jouglet et al. (2008) propose to associate a tabu status with the value of the recently visited solutions: the solutions which lead to such tabu costs are forbidden. We added this new ingredient to the proposed tabu search for (P), with the associated tabu tenure τ_5.

5 Population-based metaheuristics for (P)

In this section, we propose two adaptive memory algorithms for problem (P), which use different recombination operators. The first, dented AMA^{Mem}, uses all solutions of the memory to produce an offspring, whereas the second, denoted AMA^{SP}, uses only two solutions and the single-point crossover.

5.1 Adaptive memory algorithm AMA^{Mem}

A basic version of an AMA (Rochat and Taillard 1995) is summarized in Algorithm 1, where performing steps (1), (2), and (3) are called a *generation*. In order to design an AMA for (P), we have to define a way to initialize the population $\mathscr{M}$ of solutions, a recombination operator, an intensification (or local search) operator, and the memory update mechanism. Those elements are presented below.

First, the *population* $\mathscr{M}$ contains a set of m (parameter) solutions. $\mathscr{M}$ is initialized by generating m random ordered lists of jobs, and from such lists, resulting schedules are built

Algorithm 1 Adaptive memory algorithm

Initialize the central memory $\mathscr{M}$ with solutions.

While a stopping condition is not met, do

1. create an offspring solution s from $\mathscr{M}$ with a recombination operator;
2. apply a local search operator on s and let $s^\star$ be the resulting solution;
3. update $\mathscr{M}$ with the use of $s^\star$.

using the timing algorithm described in Sect. 4. Then, such random solutions are improved by the local search operator, and finally inserted in $\mathscr{M}$.

The *local search operator* is the tabu search method described in Sect. 4.2, which is applied during I (parameter) iterations. Since we rely on the AMA framework for bringing diversification, tabu search does not include the diversification mechanism. Note that I is a sensitive parameter, as for a given time limit, the larger is I, the less generations are performed. But the smaller is I, the less aggressive is the method. Thus, a good tradeoff has to be found.

The *recombination operator* generates an offspring solution s by considering all solutions of $\mathscr{M}$. It is formally described in Algorithm 2. Solution s is initially empty, and is built step by step. At each step, a solution of $\mathscr{M}$ is randomly chosen, and its first remaining job gives the next job to be scheduled in s. To be efficient for problem (P), a recombination operator must preserve some characteristics of the solutions of $\mathscr{M}$ to create s. Two important characteristics are (1) the chosen set of accepted jobs and (2) the order in which they are sequenced. On the one hand, the proposed recombination operator conserves the relative order of the jobs: if job j is often scheduled before job j' in $\mathscr{M}$, then job j is likely to be scheduled before job j' in s (this is always true if j and j' are copied from the same solution of $\mathscr{M}$). On the other hand, a job j which is often unperformed in the solutions of $\mathscr{M}$ will probably be unperformed in s (this is always true if j is unperformed in all solutions of $\mathscr{M}$).

The *memory update operator* relies on the following idea. If $s^\star$ (the solution provided by tabu search at the end of a generation) is strictly better than s^{worst} (the current worst solution of the population $\mathscr{M}$), then $s^\star$ replaces s^{worst}. Otherwise, $s^\star$ replaces s^{div} (the solution of $\mathscr{M}$ which is in average the most similar to the other solutions of $\mathscr{M}$; the used distance measure will be clearly defined below). Formally, remind that a solution representation $\hat{s} = (\Pi, \Omega)$ is always associated with a solution s, and s can be generated from $\hat{s}$ by the use of the timing algorithm. Given two solution representations $\hat{s}_1 = (\Pi_1, \Omega_1)$ and $\hat{s}_2 = (\Pi_2, \Omega_2)$ of solutions s_1 and s_2, the *Hamming distance* $d(s_1, s_2)$ between s_1 and s_2 computes the number of positions in Π_1 and Π_2 which do not contain the same job. The larger is $d(s_1, s_2)$, the more different solutions are s_1 and s_2. The distance measure $d_{\mathscr{M}}(s)$ computes the average Hamming distance between a solution $s \in \mathscr{M}$ and all other solutions of $\mathscr{M}$. It is defined by $d_{\mathscr{M}}(s) = \frac{1}{m-1} \cdot \sum_{s' \in \mathscr{M} - \{s\}} d(s, s')$. Thus, s^{div} is the solution minimizing $d_{\mathscr{M}}(s)$. This strategy is used to maintain some diversity in $\mathscr{M}$. Preliminary tests confirmed that this mechanism gives better results than simply replacing the oldest or worst solution of $\mathscr{M}$.

Algorithm 2 Recombination operator

Let $\hat{s} = (\Pi, \Omega)$ be the representation of the offspring solution.

Set $\Pi = \emptyset$ and $\Omega = \emptyset$.

While there remains a non-empty solution in $\mathscr{M}$, do

1. select randomly a non-empty solution representation $\hat{s}' = (\Pi', \Omega')$ in $\mathscr{M}$;
2. add the first job j of Π' at the end of the ordered list Π;
3. remove j from all solutions of $\mathscr{M}$.

Put in Ω all jobs which are not in Π.

Use the timing algorithm to build the schedule s from $\hat{s}$.

5.2 Adaptive memory algorithm AMA^{SP}

Based on the previously defined AMA^{Mem}, we propose another evolutionary strategy to tackle (P). The generation of the initial population, the local search operator, and the memory update mechanism are the same as in AMA^{Mem}. Thus, the only difference relies in the recombination operator. This operator first selects two solutions s_1 and s_2 in $\mathscr{M}$ by using a roulette wheel selection, where the fitness of each solution is related to its associated cost. The single-point crossover is applied to the sequence of performed jobs: we first select a random index κ, then the offspring sequence is composed of jobs at position 1 to κ in solution s_1, followed by jobs at position $\kappa + 1$ to the end of s_2 (except jobs which have already been taken from s_1). All jobs which are not part of the generated sequence are unperformed. From the so obtained ordered list of jobs, a schedule is built by the timing algorithm. The single-point crossover operator is illustrated in Fig. 2, where the dotted line represents the selected index κ. This crossover operator conserves the position of the jobs taken from s_1, and the relative order of the jobs taken from s_2. Also, jobs which are unperformed in both parent solutions are unperformed in the offspring solution.

6 Experiments

In this section, we compare the different algorithms presented in the paper over two sets of instances. The first subsection

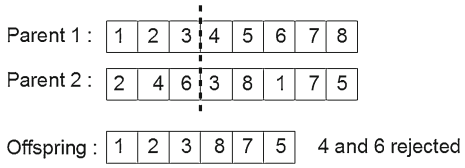

Fig. 2 Single-point crossover

presents the algorithms and indicates the used parameters. In the second subsection, we use the benchmark instances of the literature to compare our algorithms, and show that they are competitive with existing methods. In the last subsection, additional instances are introduced to conduct a more accurate comparison between the proposed algorithms, since this was not possible with the benchmark instances.

6.1 Compared algorithms

First, we indicate the tested algorithms and give the values of their associated parameters found by a preliminary tuning phase. In the purpose of having a fair comparison, we use a same time limit T (in seconds) for each method. Unless specified, T is $30 \cdot n$ seconds for each instance, where n is the number of jobs. The algorithms were implemented in C++ and ran on a computer with processor Intel i7 Quandcore (2.93 GHz RAM, 8 Go DDR3). Tests have been performed by running five times each algorithm on each instance.

The greedy algorithm (described in Sect. 4.1), denoted *Greedy*, makes some decisions randomly. More precisely, in front of two equivalent options, it break ties randomly. As a consequence, two runs of the method are very likely to result in different solutions. If *Greedy* stops before the time limit T, it is restarted from scratch as long as T is not reached. The provided solution is then the best encountered solution within T seconds. The same restarting process is used for *Descent* (the descent local search method presented in Sect. 4.2).

Tabu is the tabu search proposed in Sect. 4.2, without the diversification procedure. The used tabu tenures are $\tau_1 = 15$, $\tau_2 = 40$, $\tau_3 = 12$, $\tau_4 = 120$, $\tau_5 = 40$ for the large instances (more than 50 jobs), and $\tau_1 = 1$, $\tau_2 = 3$, $\tau_3 = 1$, $\tau_4 = 2$, $\tau_5 = 4$ for the small ones. These values have been set by a preliminary tuning phase. Note that τ_1 and τ_3 were tested in interval $[2, 20]$ for every two values, τ_2 was tested in interval $[0, 50]$ by steps of five, τ_4 was tested for every ten values in $[0, 200]$, and τ_5 was tested in $[10, 100]$ for every ten values. The tuning phase also showed that using more refined tabu tenures (e.g., dynamic instead of constant) did not improve the results. *TabuDiv* is an extension of *Tabu* by adding to it the diversification procedure.

AMA^{Mem} is the adaptive memory algorithm presented in Sect. 5.1. Preliminary experiments showed that using a population of 20 solutions, and performing tabu search for $I = 500$ iterations (with I tested in interval $[100, 10000]$), leads to the best results. AMA^{SP} is the adaptive memory algorithm with single-point crossover presented in Sect. 5.2. The used parameters are the same as in AMA^{Mem}.

Finally, *LP* denotes the linear programming formulation proposed in Sect. 2 and implemented with CPLEX 12. For this method, the time limit was fixed to one hour for all instances. *LP* was tested only on instances having at most 25 jobs, as for larger instances, the use of heuristics is necessary.

6.2 Results on the benchmark instances

In this subsection, we first describe and discuss the experiments performed with instances of the *Manufacturing Scheduling Library* (Le Pape 2007; Nuijten et al. 2004). For sake of simplicity, we call it *MaScLib* in the remainder of the paper. It is a library containing instances which are inspired from real manufacturing cases. Such benchmark instances have been made available to the research community by *ILOG*. The cost functions for each j are $f_j(C_j) = w_j \cdot T_j$, where T_j is the tardiness cost (i.e., $T_j = \max\{0, C_j - d_j\}$). Each instance contains between 8 and 500 jobs. Thirty of them do not consider setup costs or times (category NCOS), whereas fourteen assume setup costs and times (STC_NCOS). Note that instances STC_NCOS are not obtained by simply adding setup cost to NCOS instances: both types of instances were independently generated. The instances ending by a "a" assume no weight (i.e., $w_j = 1 \, \forall \, j$).

Results are presented in Table 1 (resp. Table 2) for the *MaScLib* instances without (resp. with) setups. Let $Best$ be the best solution value encountered by the set of compared methods on the considered instance, and $Result$ be the result obtain by running an algorithm on the same instance. We define the percentage Gap as $Gap = 100 \cdot \frac{Result - Best}{Best}$. For each algorithm, the average (Avg), minimum (Min) and maximum (Max) percentage gap (over five runs) are reported, with the format: Avg $(Min - Max)$. As only one run per instance is performed with LP (during a maximum of one hour), only the percentage Gap is reported.

Firstly, the tests clearly demonstrate the superiority of *Tabu* over *Descent*, and that the latter outperforms *Greedy*, since the average gaps by considering all the *MaScLib* instances (both tables) are 8.14 % for *Greedy*, 6.40 % for *Descent*, and 1.37 % for *Tabu*. Secondly, as *TabuDiv* has an average gap of 0.41 %, the diversification procedure included in *TabuDiv* is obviously useful. The gap is even more important by taking only into account instances with setups: 1.17 % for *TabuDiv* vs 4.19 % for *Tabu*, whereas it is 0.06 % for both algorithms when there are no setups. Additional informal tests confirmed that the smaller are the setups, the lower is the gap between *Tabu* and *TabuDiv*. The minimum and

Table 1 Results on the *MaScLib* instances without setups

Instance	Best	Size	LP	Greedy	Descent	Tabu	TabuDiv	AMA^{Mem}	AMA^{SP}
NCOS_01	800	8	0.00	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_01a	800	8	0.00	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_02	2570	10	0.00	12.8 (12.8–12.8)	12.8 (12.8–12.8)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_02a	1210	10	0.00	0.8 (0.8–0.8)	0.8 (0.8–0.8)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_03	6460	10	0.00	11.1 (11.1–11.1)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_03a	1690	10	0.00	3 (3–3)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_04	1011	10	0.00	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_04a	1008	10	0.00	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_05	1500	15	0.00	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_05a	1500	15	0.00	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_11	2022	20	0.00	6.1 (6.1–6.1)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_11a	2006	20	0.95	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_12	6844	24	2.73	10 (10–10)	10 (10–10)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_12a	4270	24	1.17	11.1 (11.1–11.1)	11.1 (11.1–11.1)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_13	3912	24	4.81	18.6 (18.6–18.6)	18.6 (18.6–18.6)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_13a	3441	24	17.58	9.6 (9.6–9.6)	9.6 (9.6–9.6)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_14	6990	25	51.65	1.7 (1.7–1.7)	1.7 (1.7–1.7)	0.3 (0–1.7)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_14a	3195	25	15.81	1.1 (1.1–1.1)	0.6 (0–0.9)	0.1 (0–0.6)	0.1 (0–0.6)	0 (0–0)	0 (0–0)
NCOS_15	3052	30	0.33	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_15a	3035	30	0.56	0.5 (0.5–0.5)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_31	9510	75	156.68	0 (0–0)	0 (0–0)	0.3 (0–0.8)	0.4 (0–1.1)	4.9 (4.4–5.5)	1.1 (0.7–1.9)
NCOS_31a	8715	75	95.75	4.5 (4.4–4.8)	4.7 (4.4–4.8)	0.4 (0.3–0.6)	0.7 (0.5–0.9)	0.2 (0–0.3)	0.9 (0.7–1.1)
NCOS_32	17310	75	313.06	4.2 (4.2–4.2)	4.2 (4.2–4.2)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_32a	14720	75	281.86	1.4 (1.4–1.4)	1.4 (1.4–1.4)	0.2 (0–0.3)	0.3 (0–0.3)	0.3 (0.3–0.3)	0.2 (0–0.3)
NCOS_41	13484	90	273.75	22.2 (22–22.7)	22.1 (20.9–23.5)	0.2 (0–0.4)	0.2 (0–0.3)	0.1 (0.1–0.2)	0.1 (0–0.2)
NCOS_41a	10539	90	44.23	8.2 (8–8.4)	10.3 (9.5–10.7)	0.1 (0.1–0.2)	0.1 (0–0.2)	0.1 (0–0.1)	0 (0–0.1)
NCOS_51	36170	200	*	5.8 (5.8–5.8)	4.1 (4.1–4.1)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_51a	36170	200	*	6.1 (6.1–6.1)	4.1 (4.1–4.1)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_61	1269365	500	*	0.2 (0.2–0.2)	0.2 (0.2–0.2)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
NCOS_61a	1485232	500	*	0.1 (0.1–0.1)	0.1 (0.1–0.1)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
Average				4.6 (4.6–4.7)	3.9 (3.8–4)	0.1 (0–0.2)	0.1 (0–0.1)	0.2 (0.2–0.2)	0.1 (0–0.1)

maximum gap values are close to the average for all algorithms, which indicates they are reliable. Finally, *LP* is only able to solve instances with at most 20 jobs. For the instances having between 20 and 90 jobs, *LP* is only able to give an upper bound. For larger instances, the number of variables and constraints becomes too large and the model cannot be loaded, thus no result can be found, which is indicated with a star (*) in the column *LP* of the tables.

We now compare the results obtained by *Tabu* with the upper bounds UB found in Baptiste and Pape (2005), where tests were running during 30 minutes on a 1.4 GHz PC. Since we do not use the same processor and we want the comparison to be fair, *Tabu* was stopped as soon as it obtained a solution with the same cost or a lower cost as UB. Table 3 presents the time (in seconds) needed by *Tabu* to obtain such solution values. If no time is indicated (which occurs for two instances), better or equal solutions have not been found for the corresponding instance. Except for instances *STC_NCOS_31* and *STC_NCOS_31a*, *Tabu* reaches UB very quickly. This clearly indicates that *Tabu* is a more efficient algorithm. In counterpart, the algorithm presented in Baptiste and Pape (2005) allows to demonstrate the optimality of the obtained solutions for small instances. When compared to UB, the percentage improvements ($100 \cdot \frac{UB-Result}{Result}$) are on average 8.08 % for *Greedy*, 9.67 % for *Descent*, 12.78 % for *Tabu*, 13.26 % for *TabuDiv*, 13.23 % for AMA^{Mem} and AMA^{SP}.

The performances of AMA^{SP}, AMA^{Mem} and *TabuDiv* on the *MaScLib* instances (both tables) are very similar (the gaps are, respectively, 0.37, 0.45, and 0.41 %) and do not allow a relevant comparison. We thus generate below additional instances, with more jobs.

Table 2 Results on the *MaScLib* instances with setups

Instance	Best	Size	LP	Greedy	Descent	Tabu	TabuDiv	AMA^{Mem}	AMA^{SP}
STC_NCOS_01	700	8	0.00	5.7 (5.7–5.7)	5.7 (5.7–5.7)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_01a	610	8	0.00	1.6 (1.6–1.6)	1.6 (1.6–1.6)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_15	17611	30	16.03	0.2 (0.2–0.2)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_15a	5584	30	0.38	0.1 (0.1–0.1)	0 (0–0)	0 (0–0)	0 (0–0)	1.5 (0.2–5.1)	0 (0–0)
STC_NCOS_31	6615	75	88.13	15.3 (15.1–15.4)	3.8 (3.8–3.8)	4.5 (3.8–7.6)	4.5 (3.8–7.6)	3 (0–3.8)	4.8 (3.8–7.6)
STC_NCOS_31a	7590	75	216.60	25.8 (25.6–26)	3.3 (3.3–3.3)	4.7 (3.3–6.6)	3.3 (3.3–3.3)	1.8 (0–3.3)	6.6 (4.2–8.9)
STC_NCOS_32	24068	75	70.57	4.7 (4.7–4.7)	2.8 (2.2–3)	0.3 (0–0.8)	0.9 (0.3–1.9)	1.6 (1.3–1.9)	1.3 (0.2–1.9)
STC_NCOS_32a	16798	75	174.81	3.9 (3.9–3.9)	0 (0–0)	0.5 (0–0.7)	0 (0–0)	1.2 (0.8–2.1)	0.4 (0.2–0.8)
STC_NCOS_41	43201	90	324.99	12.8 (12.8–12.9)	3.4 (3.4–3.4)	3.4 (3.3–3.4)	1.8 (0–3.4)	2.2 (0.2–3.5)	0.1 (0.1–0.1)
STC_NCOS_41a	18579	90	37.39	7.1 (7–7.1)	2.8 (2.8–2.8)	2.9 (2.8–3.4)	0.1 (0–0.4)	2.7 (2.4–3)	0.1 (0.1–0.1)
STC_NCOS_51	139675	200	*	74.7 (74.7–74.7)	74.7 (74.7–74.7)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_51a	148230	200	*	67.2 (67.2–67.2)	67.2 (67.2–67.2)	42.3 (42.3–42.3)	5.9 (2–21.3)	0 (0–0)	0.8 (0–2)
STC_NCOS_61	1495045	500	*	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0.1)
STC_NCOS_61a	1814605	500	*	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
Average				15.6 (15.6–15.7)	11.8 (11.8–11.8)	4.2 (4–4.6)	1.2 (0.7–2.7)	1 (0.3–1.6)	1 (0.3–1.6)

Table 3 Time needed by *Tabu* to reach UB

Instance	Time	Instance	Time	Instance	Time
NCOS_01	0.01	NCOS_13a	0.01	STC_NCOS_01	0.01
NCOS_01a	0.01	NCOS_14	0.01	STC_NCOS_01a	0.01
NCOS_02	0.01	NCOS_14a	13.53	STC_NCOS_15	0.01
NCOS_02a	0.01	NCOS_15	0.01	STC_NCOS_15a	0.01
NCOS_03	0.04	NCOS_15a	0.01	STC_NCOS_31	–
NCOS_03a	0.01	NCOS_31	0.01	STC_NCOS_31a	–
NCOS_04	0.01	NCOS_31a	0.01	STC_NCOS_32	0.01
NCOS_04a	0.01	NCOS_32	0.01	STC_NCOS_32a	66.99
NCOS_05	0.01	NCOS_32a	0.01	STC_NCOS_41	0.01
NCOS_05a	0.01	NCOS_41	0.01	STC_NCOS_41a	4.12
NCOS_11	0.1	NCOS_41a	4.14	STC_NCOS_51	0.03
NCOS_11a	0.01	NCOS_51	0.06	STC_NCOS_51a	0.03
NCOS_12	3.97	NCOS_51a	0.1	STC_NCOS_61	0.63
NCOS_12a	3.89	NCOS_61	0.72	STC_NCOS_61a	0.64
NCOS_13	0.01	NCOS_61a	0.54		

6.3 Results on the random instances

We have built a set of 90 random instances. A parameter α is used to control the interval of time in which release dates and due dates are generated. More precisely, a value End is randomly chosen in interval $[\sum_j p_j, (1+\alpha)\cdot\sum_j p_j]$. Then, the release date r_j of each job j is randomly picked in $[0, End - p_j]$. Its due dates d_j are randomly taken in $[r_j + p_j, End]$. One can observe that α allows to influence the rate of rejected jobs, as it controls the size of the interval of time where all jobs have to be scheduled.

The number n of jobs belongs to the set {10, 15, 20, 25, 50, 100, 200, 300, 400, 500}, and α is chosen in the set {0.5, 1, 2}. For each couple (n, α), three instances are generated (labeled with a, b, and c). The instances are named *STC_NCOS_n_α_label*. The used cost function is again $f_j(C_j) = w_j \cdot T_j$, where w_j is an integer chosen at random in [1, 5]. The abandon cost u_j of job j is related to its processing time p_j, since we believe a longer job brings more benefit to the company. Thus, we set $u_j = \beta_j \cdot p_j$, where β_j is an integer chosen at random between 10 and 60. The deadline $\bar{d}_j$ of job j is defined such that $f_j(\bar{d}_j) = u_j$. A random number of families between 10 and 20 are generated. A family is randomly assigned to each job. Setup times and costs are dependent because it seems realistic to pay a high setup cost for a long setup time (it is for instance related with the salary of employees). The setup time s_{gh} between two families g and h is an integer randomly chosen in [50, 200], and the setup cost is defined as $c_{gh} = \gamma_{gh} \cdot s_{gh}$, with $\gamma_{gh} \in [0.5, 2]$. Note that we always have $c_{gh} \leq c_{gk} + c_{kh}$ and $s_{gh} \leq s_{gk} + s_{kh}$, $\forall k$ (all the setup values which do not respect this triangle inequality are regenerated). If such a triangle inequality is not satisfied, the involved employees have better to tune the machine from state g to k, and then from state k to h, rather than directly from g to h.

Note that a different generation method was proposed in Cesaret et al. (2012), Potts and Wassenhove (1985), Akturk and Ozdemir (2000). The method uses two parameters to define the instances: the average tardiness factor and the due dates range. We propose here to use a single parameter α controlling the size of the time window in which all jobs have to be scheduled. It thus allows to directly have an influence on the rate of accepted jobs in a feasible solution. Note that α has

Table 4 Results on small size random instances

Instance	Best	LP	Greedy	Descent	Tabu	TabuDiv	AMA^{Mem}	AMA^{SP}
STC_NCOS_10_0.5_a	13315	0.00	9.5 (9.5–9.5)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_10_0.5_b	19536	0.00	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_10_0.5_c	12199	0.00	4 (4–4)	1.2 (1.2–1.2)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_10_1_a	8344	0.00	26.7 (26.7–26.7)	26.7 (26.7–26.7)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_10_1_b	14492	0.00	0.8 (0.8–0.8)	0.8 (0.8–0.8)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_10_1_c	10905	0.00	11.8 (11.8–11.8)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_10_2_a	12435	0.00	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_10_2_b	22306	0.00	5.3 (5.3–5.3)	5.3 (5.3–5.3)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_10_2_c	11777	0.00	3.9 (3.9–3.9)	3.9 (3.9–3.9)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_15_0.5_a	21924	0.86	17.5 (17.5–17.5)	14.2 (14.2–14.2)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_15_0.5_b	38240	6.12	11.8 (11.8–11.8)	2.1 (2.1–2.1)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_15_0.5_c	33316	4.00	1.9 (1.9–1.9)	1.5 (1.5–1.5)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_15_1_a	14368	2.16	37.1 (37.1–37.1)	24.3 (24.3–24.3)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_15_1_b	23752	0.08	0.3 (0.3–0.3)	0.3 (0.3–0.3)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_15_1_c	29952	1.48	3.4 (3.4–3.4)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_15_2_a	23779	4.88	7.9 (7.9–7.9)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_15_2_b	11881	5.71	11.7 (11.7–11.7)	10.5 (10.5–10.5)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_15_2_c	11252	7.34	42.1 (42.1–42.1)	0.3 (0.3–0.3)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_20_0.5_a	64907	13.36	6.8 (6.8–6.8)	6.8 (6.8–6.8)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_20_0.5_b	54313	2.82	0.8 (0.8–0.8)	0.8 (0.8–0.8)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_20_0.5_c	58346	1.51	8.2 (8.2–8.2)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_20_1_a	63399	1.50	15.4 (15.4–15.4)	1.5 (1.5–1.5)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_20_1_b	42789	5.18	2.2 (2.2–2.2)	0.8 (0.8–0.8)	0.4 (0.4–0.4)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_20_1_c	48019	0.92	0.2 (0.2–0.2)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_20_2_a	47358	55.47	6.4 (6.4–6.4)	6.2 (6.2–6.2)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_20_2_b	55484	0.59	17.8 (17.8–17.8)	16.6 (16.6–16.6)	2.8 (0–7.7)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_20_2_c	14883	50.93	29.1 (29.1–29.1)	7.4 (7.4–7.4)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_25_0.5_a	55749	4.74	8.3 (8.3–8.3)	7.2 (7.2–7.2)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_25_0.5_b	59758	10.27	21.4 (21.4–21.4)	7.9 (7.9–7.9)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_25_0.5_c	60756	2.21	18.5 (18.5–18.5)	4.5 (4.5–4.5)	4.5 (4.5–4.5)	0 (0–0)	0.2 (0–0.5)	0 (0–0)
STC_NCOS_25_1_a	62302	12.51	21.3 (21.3–21.3)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_25_1_b	54952	8.90	32.5 (32.5–32.5)	9.1 (9.1–9.1)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_25_1_c	67897	3.30	19.7 (19.7–19.7)	4.4 (4.4–4.4)	1.6 (1.6–1.6)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_25_2_a	16484	24.97	53.7 (53.7–53.7)	25 (23.5–30.9)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_25_2_b	29921	32.56	35.9 (35.9–35.9)	1 (1–1)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
STC_NCOS_25_2_c	6807	36.30	56.4 (56.4–56.4)	3.6 (0–8.9)	0 (0–0)	0 (0–0)	0 (0–0)	0 (0–0)
Average		8.4	15.3 (15.3–15.3)	5.4 (5.2–5.7)	0.3 (0.2–0.4)	0 (0–0)	0 (0–0)	0 (0–0)

also a direct impact on the average tardiness. In contrast with other papers, we do not control the slack time value, neither the range of due dates (for a problem similar to (P); Oğuz et al. (2010) observed no influence of this latter parameter on the results).

We first present results for small size instances in Table 4. Except for *LP*, five runs were performed on each instance by each algorithm. *LP* is only able to solve all instances with 10 jobs, and allows to obtain good upper bounds for instances with up to 20 jobs. The average gap obtained by *LP* is greater for instances generated with $\alpha = 200$, which shows that those instances are harder: the average gaps are, respectively, 3.82, 3.00, and 18.23 % when α is equal to 50, 100, and 200. As it was the case for the *MaScLib* instances, *Tabu* outperforms *Descent*, and the latter is better than *Greedy*, since their average gaps are, respectively, 0.3, 5.4, and 15.3 %. *TabuDiv* and AMA^{SP} obtain best results for all instances and all runs. Excepts for instance STC_25_0.5c, AMA^{Mem} also gets the best results.

Table 5 Results on large size random instances

Instance	Best	Greedy	Descent	Tabu	TabuDiv	AMA^{Mem}	AMA^{SP}
STC_NCOS_50_0.5_a	32517	51.2 (51.2–51.2)	11.9 (10.4–15.4)	0.1 (0–0.3)	0.2 (0–0.4)	7.9 (6.7–8.9)	0.4 (0.1–0.9)
STC_NCOS_50_0.5_b	166969	25.3 (25.3–25.3)	7 (7–7)	9 (6.6–9.7)	0 (0–0)	0.4 (0.2–1.1)	0 (0–0)
STC_NCOS_50_0.5_c	258505	9.8 (9.8–9.8)	7.6 (5.6–8.9)	5.6 (5.6–5.6)	0 (0–0)	0.1 (0–0.1)	0 (0–0)
STC_NCOS_50_1_a	22387	120.2 (120.2–120.2)	48 (38.4–54.4)	2.5 (1.3–3.8)	3.3 (2.4–5.1)	19.9 (17.8–21.5)	1.2 (0–3.2)
STC_NCOS_50_1_b	102420	34.8 (34.8–34.8)	20.1 (13.1–21.8)	10.2 (6.7–17.2)	0 (0–0)	0.5 (0.2–1.4)	0.3 (0.1–1.1)
STC_NCOS_50_1_c	75698	36.6 (36.6–36.6)	7.3 (7.3–7.3)	6.9 (6.9–6.9)	0 (0–0)	1.4 (0.7–2.1)	0.7 (0.2–1.2)
STC_NCOS_50_2_a	28065	81.7 (81.7–81.7)	19.2 (16.3–25.3)	1.2 (0.6–1.8)	1.1 (0.7–1.7)	12.6 (10.9–14.4)	0.1 (0–0.4)
STC_NCOS_50_2_b	23868	87.3 (87.3–87.3)	44.8 (35.4–49.1)	28.1 (13.4–37)	0 (0–0)	1.1 (0.3–1.6)	0.1 (0–0.3)
STC_NCOS_50_2_c	36416	171.6 (171.6–171.6)	24.4 (16.3–34.4)	4.8 (2.8–6.6)	0 (0–0)	5.9 (3.9–8.6)	1.5 (0.1–2.2)
STC_NCOS_100_0.5_a	59696	113.5 (113.5–113.5)	37 (30.3–41.3)	0.2 (0–0.5)	1.2 (0.4–1.5)	8.7 (7.8–9.8)	3.2 (1.9–3.9)
STC_NCOS_100_0.5_b	811040	17.2 (17.2–17.2)	7.8 (7.2–7.9)	5 (4.7–5.3)	0 (0–0.1)	0.8 (0.6–1)	0.1 (0–0.3)
STC_NCOS_100_0.5_c	551612	29.7 (29.7–29.7)	16 (14.3–17)	12 (9–16.5)	0.2 (0–0.4)	1 (0.4–1.6)	0.3 (0.2–0.7)
STC_NCOS_100_1_a	8805	192.3 (192.3–192.3)	31.1 (16.2–46.3)	10.7 (7.1–12.6)	5.8 (0–11.7)	39.2 (34.7–41.9)	15.6 (12.5–17.5)
STC_NCOS_100_1_b	202646	75 (75–75)	20.2 (17.6–22.4)	10.2 (2.3–17.1)	1.6 (0.3–2.3)	2.3 (1.2–3.5)	0.8 (0–1.2)
STC_NCOS_100_1_c	414155	33.7 (33.7–33.7)	15.8 (15.2–17.1)	10.3 (8.9–13.3)	0.7 (0–1.5)	3.6 (3.1–4.4)	0.8 (0.3–1)
STC_NCOS_100_2_a	6208	152.7 (152.7–152.7)	15.1 (0.7–31.4)	6.8 (1.7–10.1)	4.8 (0–10)	28.5 (23–34.4)	11 (8–15.7)
STC_NCOS_100_2_b	37284	165 (164.4–165.8)	45.6 (14.2–93.5)	7.6 (4.7–10.1)	13.5 (9.1–18.7)	10.1 (7.2–12.1)	2.8 (0–5.9)
STC_NCOS_100_2_c	305176	65.1 (65.1–65.1)	25.1 (17.7–29.2)	12 (8–15.4)	1 (0–1.8)	3.1 (1.9–4.2)	1.3 (0–2.9)
STC_NCOS_200_0.5_a	105154	137.6 (135.4–138.9)	22 (14.1–27.2)	3.4 (1–5.2)	2.1 (0.3–3.3)	15.3 (14.6–15.7)	3.1 (0–6.8)
STC_NCOS_200_0.5_b	1233620	43.7 (43.7–43.7)	15.5 (12.1–17.2)	11.4 (8.6–14.7)	2.5 (2.1–2.7)	4.9 (4.2–5.5)	1.3 (0–2.6)
STC_NCOS_200_0.5_c	2322007	16.8 (16.8–16.8)	4.8 (4.8–4.8)	4.7 (4.3–4.8)	0.5 (0–1)	1.4 (1.2–1.7)	0.5 (0.2–0.7)
STC_NCOS_200_1_a	22796	175.5 (173.9–176.5)	15.2 (9–28.2)	2.6 (0–5.4)	1.7 (0.6–3.3)	26.7 (22.9–30.4)	15 (9.3–21.2)
STC_NCOS_200_1_b	1023537	72.4 (72.4–72.4)	17.1 (10.9–20.2)	18.4 (12.1–22.2)	0.4 (0–1.1)	6.9 (4.9–7.5)	1.6 (0.8–2.8)
STC_NCOS_200_1_c	2312750	17.4 (17.4–17.4)	3.7 (3–4.2)	3.2 (3–4.2)	0.5 (0.1–1)	0.3 (0–0.5)	0.5 (0.4–0.7)
STC_NCOS_200_2_a	12662	59.5 (58–60.5)	1.9 (−0.3–3.8)	1.7 (0–2.6)	2.8 (2.2–3.2)	21.6 (20.2–22.5)	13.5 (11.6–15.1)
STC_NCOS_200_2_b	679767	101.4 (101.4–101.5)	27.4 (16.2–39.6)	22.8 (13.4–32.3)	3.5 (1.1–6.5)	10.3 (7.6–12.2)	3.9 (0–7.4)
STC_NCOS_200_2_c	1374460	49.9 (49.9–49.9)	13.7 (10.3–17.5)	13.5 (10.4–17.8)	1 (0–2.1)	3.7 (2.7–4.7)	0.9 (0–1.7)
STC_NCOS_300_0.5_a	134922	158.2 (156.7–159.7)	28.6 (22.6–33.4)	9.3 (6–12.5)	4.2 (1.1–5.7)	21.3 (18.9–22.5)	2.4 (0–4.1)
STC_NCOS_300_0.5_b	3628402	20.9 (20.9–20.9)	4.2 (4.2–4.2)	5.5 (4.6–6.1)	0.3 (0–0.8)	1.5 (1.2–1.9)	0.5 (0.3–0.6)
STC_NCOS_300_0.5_c	3674540	12.8 (12.8–12.8)	1.6 (1.6–1.7)	1.6 (1.6–1.7)	0.2 (0.1–0.5)	1.2 (0–1.7)	0.5 (0–1)
STC_NCOS_300_1_a	19529	415.2 (395.4–444.9)	29.4 (13.1–43.9)	10.9 (0–34.8)	5 (3.4–8.2)	69.5 (65–73)	17.6 (15.3–20.4)
STC_NCOS_300_1_b	1204038	97.4 (96.3–98.5)	16.6 (14.2–17.7)	15.4 (12.4–20.7)	1.5 (0–3.2)	6.2 (3.5–7.7)	2.8 (1.9–3.3)
STC_NCOS_300_1_c	1585766	77.1 (77.1–77.1)	15.4 (12.8–17.9)	15.4 (9.9–20.7)	1 (0–2.9)	9.7 (7.9–10.7)	1.4 (0–4.6)
STC_NCOS_300_2_a	16741	78.4 (77.7–79.1)	11.7 (4.7–16.4)	2.1 (1.4–2.6)	3.1 (0–6.7)	32.4 (29.7–34.9)	13.5 (4.7–18.9)
STC_NCOS_300_2_b	300466	300.3 (290.1–305)	50.9 (37.2–68.4)	40.4 (32.6–49.6)	6.7 (0–15.7)	49.3 (44.5–52.7)	17.1 (11.1–24.1)

Table 5 continued

Instance	Best	Greedy	Descent	Tabu	TabuDiv	AMA^{Mem}	AMA^{SP}
STC_NCOS_300_2_c	4236180	16.5 (16.5–16.5)	2.5 (2.1–3.2)	2.8 (2.1–3.2)	1 (0.6–1.4)	0.8 (0.6–1)	0.5 (0–0.8)
STC_NCOS_400_0.5_a	173621	143.1 (136.4–145.1)	19 (14.2–22.6)	10.5 (4.7–17.5)	6.5 (4.9–7.1)	27.5 (26.4–29.1)	1.9 (0–3.5)
STC_NCOS_400_0.5_b	5259270	15.3 (15.3–15.3)	4.6 (2.6–5.2)	4 (2.5–5.2)	1.1 (0.7–1.3)	2.4 (1.9–2.8)	0.7 (0–1.3)
STC_NCOS_400_0.5_c	4949009	15.2 (15.2–15.2)	3.5 (3.1–3.7)	3.2 (2.9–3.5)	0.6 (0–0.9)	1.9 (1.7–2.1)	1.2 (1–1.6)
STC_NCOS_400_1_a	63699	287.8 (283.3–292.3)	31.1 (22.8–48.6)	14.8 (7.1–22.8)	1.2 (0–2.7)	55.1 (49.6–58.8)	11.6 (4–18.8)
STC_NCOS_400_1_b	3294860	46.5 (46.2–46.8)	10.1 (8.7–12.1)	10.4 (8.6–12.6)	1.3 (0.4–2.2)	3.3 (0–4.7)	1.9 (0.9–2.8)
STC_NCOS_400_1_c	3523737	47.4 (47.3–47.5)	9.5 (7.9–11.2)	9.6 (6.5–11.4)	0.7 (0–1.3)	3.2 (2.4–3.6)	1.8 (1.4–2.4)
STC_NCOS_400_2_a	10177	23.6 (23.6–23.6)	8.4 (5.1–11)	1.4 (0–3.1)	3.9 (2.4–5.9)	53.3 (51.5–56.1)	19.6 (17.2–21.7)
STC_NCOS_400_2_b	3505100	44.2 (44–44.3)	6.8 (5.6–8.1)	8.5 (7.6–9.5)	0.9 (0.1–1.2)	4.1 (2.6–5)	1.7 (0–2.8)
STC_NCOS_400_2_c	64548	634.1 (618.2–643.5)	67.8 (44.9–116.9)	53.6 (25.4–86.2)	5.1 (0–8.5)	209.7 (177.1–243.7)	36.4 (24.3–47.5)
STC_NCOS_500_0.5_a	203806	166.4 (165–167.5)	16.5 (8–20.3)	7.4 (1.3–13.7)	7.4 (4–9.6)	35.4 (31–37.2)	1 (0–1.9)
STC_NCOS_500_0.5_b	5380680	21.3 (20.4–21.7)	3.3 (2.3–4.7)	3.5 (1.8–4.8)	0.4 (0–0.9)	1.8 (1–2.6)	0.7 (0.1–1.1)
STC_NCOS_500_0.5_c	5730160	26.1 (25.9–26.2)	2.7 (1.1–4.4)	2.9 (1.3–4.8)	0.5 (0.3–0.7)	1.1 (0.7–1.5)	0.7 (0–1.6)
STC_NCOS_500_1_a	234139	145.1 (141.5–146.8)	17.2 (15.2–18.2)	4.7 (0.2–8.9)	2.9 (2.3–3.7)	27 (25.6–28.2)	0.7 (0–2.4)
STC_NCOS_500_1_b	2971787	79.4 (79.2–79.6)	10.7 (6.2–14.2)	10.3 (7.1–12.5)	0.7 (0–1.8)	11 (10.5–11.8)	3.7 (0.5–7.1)
STC_NCOS_500_1_c	2358376	94.1 (92.7–95.6)	6.1 (2.4–8.7)	7.2 (5.7–9.2)	1.6 (0–3.2)	13.4 (12–14.5)	2.7 (1.2–4.8)
STC_NCOS_500_2_a	14359	30.8 (30.6–31.2)	9.1 (4–11.9)	3.8 (0–9.4)	2 (0–3.2)	44.9 (42.3–47.7)	12.2 (8.8–14.4)
STC_NCOS_500_2_b	367857	400.2 (394.8–403.5)	32 (22–40.1)	21.7 (12.4–32.8)	7 (0–19.8)	124.8 (120–129.2)	23 (11.2–34.6)
STC_NCOS_500_2_c	1928830	115.9 (115.2–116.5)	9.1 (5.3–13.5)	9.4 (8.6–10.5)	3 (1.1–5.8)	19.2 (17.3–20)	4.3 (0–7.2)
Average		104.6 (103.1–105.9)	17.7 (12.2–23.6)	9.5 (5.9–13.5)	2.2 (0.8–3.8)	19.8 (17.5–21.8)	4.9 (2.8–6.9)

Table 5 presents the results, as defined before, on large size instances for all the presented algorithms. *LP* was not tested on such instances, as its limited performance was already showed on the *MaScLib* instances. Each algorithm has been run five times on each instance. On this set of instances, results confirm that *Tabu* is better than *Descent*, and the latter is better than *Greedy*, since their average gaps are, respectively, 9.5, 17.7, and 104.6 %. Regarding the results on all the instances (i.e., associated with the four tables), it can be noticed that *Greedy* often gives the same results for the five runs (as the minimum and maximum percentage gaps are equal). This can be explained by the restarting process of the method. Depending on random decisions (due to ties occurring during the job ordering phase or during the choice of a position to insert each job), two runs of the constructive heuristic are likely to produce different solutions. However, the number of equivalent solutions is small in comparison with the number of generated solutions within an hour. Regarding the random instances, such equalities usually occur on the small instances, since with larger instances, the number of random decisions increases and the number of generated solutions within the time limit decreases. The same kind of behavior holds for *Descent*.

When comparing AMA^{SP} and AMA^{Mem}, the results show that AMA^{SP} performs better. The average gap is 4.9 % for AMA^{SP}, whereas it is 19.8 % for AMA^{Mem}. We can thus conclude that for (P), it is preferable to use AMA^{SP} rather than AMA^{Mem}. We give three ideas which could explain the disappointing behavior of AMA^{Mem}. Firstly, on the contrary to AMA^{SP}, the AMA^{Mem} recombination operator does not conserve jobs adjacency (i.e., a job is not likely to have the same neighbors as one of its parents). A second weakness could be the use of all the memory to build an offspring solution, instead of only two parent solutions. The recombination operator defined in AMA^{Mem} conserves the relative order of jobs: two jobs j and k copied from the same solution will have the same order in the offspring solution. The more there are parent solutions involved to build an offspring solution, the less such relations are maintained, since less jobs are taken from the same solution. As a consequence, we can assume that trying to recombine too many solutions is not likely to capture relevant information contained in the parents. A third drawback is that when the offspring solution is built, more importance is given to the jobs which are at the beginning of the parents, as the first not already added job of the selected parent is added to the offspring solution.

The difference gap between *Tabu* and *TabuDiv* (around 7 %) outlines again that the diversification mechanism designed for tabu search is suitable for (P). *TabuDiv* is an efficient metaheuristic which outperforms on average the other proposed algorithms. Regarding all instances, population-based methods are not as powerful. It can, however, be noticed that AMA^{SP} performs slightly better on instances for which a lot of jobs have to be rejected (i.e., with $\alpha = 0.5$). AMA^{SP} often obtains the best performance on such instances: its average gap is 0.62, and 0.93 % for *TabuDiv*. But *TabuDiv* performs better on other instances.

7 Conclusion

In this paper, we studied the one machine scheduling problem (P) introduced in Baptiste and Pape (2005), which involves release dates, setups, deadlines, and the possibility to reject some jobs. To the best of our knowledge, there is no other literature on (P), which is, however, relevant in practice. An integer linear formulation, a greedy heuristic, and a tabu search are first presented. We show that tabu search is competitive, and that the proposed diversification technique is useful. Two adaptive memory algorithms are also proposed. Tests show that the method using the single-point crossover (with two parent solutions) is more efficient than generating an offspring with all the solutions of the memory. On the studied set of instances, the tabu search approach is on average more powerful. However, the adaptive memory algorithm allows to improve tabu search when more jobs are likely to be rejected.

Future works are two folds: the design of other methods, and the study of extensions of (P). Among the possible new metaheuristics for the problem, we can be interested in methods mimicking social behaviors (e.g., particle swarm optimization, ant system algorithms), or in the use of the proposed neighborhoods within a variable neighborhood search framework (which means that the neighborhoods would be sequentially used, instead of jointly). Among the possible extensions of (P), we could use non-regular cost functions which are often encountered in a just-in-time environment (e.g., with weighted earliness and tardiness penalties), and we could tackle some additional constraints (e.g., precedence constraints). We can finally mention the consideration of the same types of costs, but with several machines.

References

Akturk, M. S., & Ozdemir, D. (2000). An exact approach to minimizing total weighted tardiness with release dates. *IIE Transactions*, *32*, 1091–1101.

Akturk, M. S., & Ozdemir, D. (2001). A new dominance rule to minimize total weighted tardiness with unequal release dates. *European Journal of Operational Research*, *135*(2), 394–412.

Anghinolfi, D., & Paolucci, M. (2007). Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. *Computers & Operations Research*, *34*(11), 3471–3490.

Baptiste, P., & Le Pape, C. (2005). Scheduling a single machine to minimize a regular objective function under setup constraints. *Discrete Optimization*, *2*(1), 83–99.

Bilgintürk Yalçın, Z., Oğuz, C., & Salman Sibel, F. (2007). Order acceptance and scheduling decisions in make-to-order systems. In *Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Application (MISTA 2007)* (pp. 80–87). Paris.

Bo, L., Ling, W., Ying, L., & Shouyang, W. (2011). A unified framework for population-based metaheuristics. *Annals of Operations Research*, *186*(32), 231–262.

Bożejko, W. (2010). Parallel path relinking method for the single machine total weighted tardiness problem with sequence-dependent setups. *Journal of Intelligent Manufacturing*, *21*, 777–785.

Cesaret, B., Oğuz, C., & Sibel Salman, F. (2012). A tabu search algorithm for order acceptance and scheduling. *Computers & Operations Research*, *39*(6), 1197–1205. Special Issue on Scheduling in Manufacturing Systems.

Du, J., & Leung, J. Y. (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, *15*, 483–495.

Gendreau, M., & Potvin, J.-Y. (2010). *Handbook of Metaheuristics* (2nd ed.). New York: Springer.

Goslawski, M., Józefowska, J., Kulus, M., & Nossack, J. (2014). Scheduling orders with mold setups in an injection plant. In *Proceedings of the 14th International Workshop on Project Management and Scheduling, PMS 2014*, Munich, Germany.

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

Harrison, S.A., Philpott, M.S., & Price, M.E. (1999). Task scheduling for satellite based imagery. In *Proceedings of the 18th Workshop of the UK Planning and Scheduling Special Interest Group* (pp. 64–78) Salford: University of Salford.

Hertz, A., & Kobler, D. (2000). A framework for the description of evolutionary algorithms. *European Journal of Operational Research*, *126*, 1–12.

Hsu, H. M., Hsiung, Y., Chen, Y. Z., & Wu, M. C. (2009). A GA methodology for the scheduling of yarn-dyed textile production. *Expert Systems with Applications*, *36*(10), 12,095–12,103.

Jouglet, A., Savourey, D., Carlier, J., & Baptiste, P. (2008). Dominance-based heuristics for one-machine total cost scheduling problems. *European Journal of Operational Research*, *184*(3), 879–899.

Kellegöz, T., Toklu, B., & Wilson, J. (2008). Comparing efficiencies of genetic crossover operators for one machine total weighted tardiness problem. *Applied Mathematics and Computation*, *199*(2), 590–598.

Kirlik, G., & Oğuz, C. (2012). A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine. *Computers and Operations Research*, *39*(7), 1506–1520.

Laguna, M., Barnes, J. W., & Glover, F. (1991). Tabu search methods for a single machine scheduling problem. *Journal of Intelligent Manufacturing*, *2*, 63–73.

Le Pape, C. (2007). A Test Bed for Manufacturing Planning and Scheduling Discussion of Design Principles. In *Proceedings of the International Workshop on Scheduling a Scheduling Competition, Providence Rhode Island USA*.

Lee, Y. H., Bhaskaran, K., & Pinedo, M. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, *29*(1), 45–52.

Lemaitre, M., Verfaillie, G., Jouhaud, F., Lachiver, J. M., & Bataille, N. (2002). Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology*, *6*, 367–381.

Lü, Z., Glover, F., & Hao, J.K. (2009). Neighborhood combination for unconstrained binary quadratic problems. In *Proceedings of the 8th Metaheuristic International Conference*. Hamburg, Germany, 13–16 July, 2009.

Nagarur, N., Vrat, P., & Duongsuwan, W. (1997). Production planning and scheduling for injection moulding of pipe fittings a case study. *International Journal of Production Economics*, *53*(2), 157–170.

Nobibon, F. T., & Leus, R. (2011). Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment. *Computers & Operations Research*, *38*, 367–378.

Nobibon, F.T., Herbots, J., & Leus, R. (2009). Order acceptance and scheduling in a single-machine environment: Exact and heuristic algorithms. In *Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2009), Dublin, Ireland*.

Nuijten, W., Bousonville, T., Focacci, F., Godard, D., & Le Pape, C. (2004). Towards an industrial manufacturing scheduling problem and test bed. In *Proceedings of Project Management and Scheduling (PMS 2004), Nancy* (pp. 162–165).

Oğuz, C., Sibel Salman, F., & Bilgintürk Yalçın, Z. (2010). Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics*, *125*(1), 200–211.

Pinedo, M. (2008). *Scheduling: Theory, algorithms, and systems*. Berlin: Springer.

Potts, C. N., & van Wassenhove, L. N. (1985). A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, *33*(2), 363–377.

Ribeiro, F., de Souza, S., Souza, M., & Gomes, R. (2010). An adaptive genetic algorithm to solve the single machine scheduling problem with earliness and tardiness penalties. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 1–8).

Rochat, Y., & Taillard, E. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, *1*, 147–167.

Rom, W. O., & Slotnick, S. A. (2009). Order acceptance using genetic algorithms. *Computers & Operations Research*, *36*(6), 1758–1767.

Sels, V., & Vanhoucke, M. (2011). A hybrid dual-population genetic algorithm for the single machine maximum lateness problem. *Lecture Notes in Computer Science*, *6622*, 14–25.

Shabtay, D., Gaspar, N., & Yedidsion, L. (2012). A bicriteria approach to scheduling a single machine with job rejection and positional penalties. *Journal of Combinatorial Optimization*, *23*(4), 395–424.

Shin, H. J., Kim, C. O., & Kim, S. S. (2002). A tabu search algorithm for single machine scheduling with release times, due dates, and sequence-dependent set-up times. *The International Journal of Advanced Manufacturing Technology*, *19*, 859–866.

Slotnick, S. A. (2011). Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research*, *212*(1), 1–11.

Taillard, E. D., Gambardella, L. M., Gendreau, M., & Potvin, J. Y. (2001). Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, *135*, 1–16.

Thevenin, S., Zufferey, N., & Widmer, M. (2012). Tabu search for a single machine scheduling problem with rejected jobs, setups and deadlines. In *9th International Conference of Modeling, Optimization and SIMulation (MOSIM 2012), Bordeaux*.

Vepsalainen, A. P. J., & Morton, T. E. (1987). Priority rules for job shops with weighted tardiness costs. *Management Science*, *33*(8), 1035–1047.

Wei-Cheng, L., & Chang, S. C. (2005). Hybrid algorithms for satellite imaging scheduling. *Systems, Man and Cybernetics, Hawaii, USA*, *3*, 2518–2523.

Wu, Q., Hao, J. K., & Glover, F. (2012). Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, *196*, 611–634.

Xiao, Y. Y., Zhang, R. Q., Zhao, Q. H., & Kaku, I. (2012). Permutation flow shop scheduling with order acceptance and weighted tardiness. *Applied Mathematics and Computation*, *218*(15), 7911–7926.

Yang, B., & Geunes, J. (2007). A single resource scheduling problem with job-selection flexibility, tardiness costs and controllable processing times. *Computers & Industrial Engineering*, *53*(3), 420–432.

Zhang, L., Lu, L., & Yuan, J. (2009). Single machine scheduling with release dates and rejection. *European Journal of Operational Research*, *198*(3), 975–978.

Zorzini, M., Corti, D., & Pozzetti, A. (2008). Due date (dd) quotation and capacity planning in make-to-order companies: Results from an empirical analysis. *International Journal of Production Economics*, *112*(2), 919–933.

Zufferey, N. (2012). Metaheuristics: Some principles for an efficient design. *Computer Technology and Application*, *3*, 446–462.

Zufferey, N., Amstutz, P., & Giaccari, P. (2008). Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling*, *11*(4), 263–277.