An exact approach for scheduling jobs with regular step cost functions on a single machine

Boris Detienne[a], Stéphane Dauzère-Pérès[b], Claude Yugma[b]

[a]Université d'Avignon et des pays de Vaucluse
339, chemin des Meinajaries
Agroparc BP 1228
84 911 Avignon Cedex 9 France
[b] École des Mines de Saint-Étienne
Centre Microélectronique de Provence, Site Georges Charpak
880 route de Mimet, 13541 Gardanne, France

Abstract

This paper studies a single machine scheduling problem whose objective is to minimize a regular step total cost function. Lower and upper bounds, obtained from linear and Lagrangean relaxations of different Integer Linear Programming formulations, are compared. A dedicated exact approach is presented, based on a Lagrangean relaxation. It consists of finding a Constrained Shortest Path in a specific graph designed to embed a dominance property. Filtering rules are developed for this approach in order to reduce the size of the graph, and the problem is solved by successively removing infeasible paths from the graph. Numerical experiments are conducted to evaluate the lower and upper bounds. Moreover, the exact approach is compared with a standard solver and a naive branch-and-bound algorithm.

Keywords:
One machine scheduling, exact method, Lagrangean relaxation, Lagrangean heuristic, total cost minimization, step cost function.

[☆]Part of this work was supported by STMicroelectronics and the research project Rousset 2003-2008, financed by the Communauté du Pays d'Aix, Conseil Général des Bouches du Rhône and Conseil Régional Provence Alpes Côte d'Azur.

Email addresses: `boris.detienne@univ-avignon.fr` (Boris Detienne), `dauzere-peres@emse.fr` (Stéphane Dauzère-Pérès), `yugma@emse.fr` (Claude Yugma)

1. Introduction

Scheduling can be broadly defined as the process of allocating resources and time intervals to tasks (or jobs) so that one or more criteria are optimized. A number of objective functions have been investigated in the literature, most of them classified among *bottleneck objectives* or *sum objectives* (Brucker, 2004). The former consist of the minimization of the maximum evaluation of a criterion over all tasks, such as minimizing the Makespan (C_{max}) or the maximum lateness (L_{max}). In the later, each task of the schedule is assigned a cost depending on its completion time. The aim is then to minimize the sum of the costs of all tasks. These problems comprise the minimization of the total (weighted) tardiness ($\sum_j w_j T_j$), the minimization of the (weighted) number of tardy jobs ($\sum_j w_j U_j$) or the minimization of earliness and tardiness penalties ($\sum_j \alpha_j E_j + \beta_j T_j$).

In this paper, we are interested in a problem which generalizes the minimization of the weighted number of tardy jobs. Let us recall that, in the classical $\sum_j w_j U_j$ problem, each job has a due date d_j and, if a job j is completed after d_j, a cost w_j is incurred that is independent of the completion time of job j. The cost w_j represents for example the loss of a customer order or good-will, raw materials being perished, financial penalties ... In this paper, we assume that each job has a set of due dates and a set of associated increasing costs. It allows the modeling of the same real-life phenomena while introducing a gradation on the importance of being late. This is for instance necessary when multiple discrete deadlines can be proposed to the customers with different preferences. Stepwise tardiness costs arise in various applications: railroad scheduling (Sahin, 2006), truckload, expedited ground, airfreight forwarding (Curry and Peters, 2005) or when considering transshipment fees which increase along with job completion times in scheduling problems (Yang, 2009). Another practical application in the context of semiconductor manufacturing that originally motivated this research is described in (Detienne et al., 2009). Lots of different product types are manufactured with a given production schedule, and the goal is to schedule some of these lots on one or several measurement machines to minimize the risks. In this case, the risk w_j corresponds to the number of lots of the same product type that will be started if the measurement of a lot is completed after a due date d_j, and thus may be lost if a quality problem is found. Each lot to be measured has thus a set of due date and associated risks, corresponding to the start times of the lots of the same product type in the production schedule.

This paper focuses on the single-machine case of the problem with regular cost functions, which is described in Section 1.1, and aims at comparing the effectiveness of different approximate and exact methods. Related work found in the literature is discussed in Section 1.2. Integer Linear Programming (ILP) formulations are presented in Section 1.3, and Section 1.4 studies the complexity of the problem. Sections 2.1 and 2.2 respectively propose Lagrangean lower and upper bounds. An exact method is described in Section 3. Numerical results on the Lagrangean heuristics and the exact approach are reported and discussed in Section 4. Some conclusions and perspectives are given in Section 5.

1.1. Problem formulation

Our problem consists in scheduling a set of jobs on a single machine, which can handle at most one job at a time. The jobs and the machine are not subject to availability constraints, and the objective is to minimize a regular step cost function of the job completion times. Formally, an instance of our problem is determined by:

- a set J of jobs. For each job $j \in J$:
- a processing time p_j,
- an ordered set $K_j = (d_j^1, \ldots, d_j^{s_j})$ of s_j jump points,
- and an ordered set $\Gamma_j = (\gamma_j^1, \ldots, \gamma_j^{s_j})$ of costs, that describe the cost function of job j, such that $\gamma_j^1 < \gamma_j^2 < \cdots < \gamma_j^{s_j}$, $j \in J$.

A cost γ_j^l is incurred if job j completes at time t such that $d_j^l < t \leq d_j^{l+1}$, $l \in \{1, \ldots, s_j - 1\}$. A cost $\gamma_j^{s_j}$ is incurred if $t > d_j^{s_j}$. Without loss of generality, we assume that no cost is incurred if $t \leq d_j^1$. If necessary, data can be modified by adding a constant term to the objective function and adapting the costs consequently. Our study considers the case of regular cost functions, i.e. $\gamma_i^1 < \gamma_i^2 < \cdots < \gamma_j^{s_j}$, $j \in J$. Let $\eta = \sum s_j$ denote the total number of jump points, and $D = \cup_{j \in J} K_j$ denote the set of distinct jump points. Note that $|D| \leq \eta$ since several jump points of different jobs might be equal.

The problem is to assign a starting time to each job, such that the cost of the schedule is minimum. Jobs are scheduled without preemption, and all data is integer. This problem can be denoted, in the standard three field notation (Graham et al., 1979), by $1||\sum \bar{f}_j(C_j)$ where $\bar{f}_j$ is a non-decreasing step function.

1.2. State of the art

To our knowledge, only a few papers about machine scheduling deal explicitly with a regular step cost function (with the exception of the criterion $\sum_j w_j U_j$). (Yang, 2009) calls this problem the *multiple due dates problem* and focuses on the special case where all jobs have the same set of jump points. The author designs a branch-and-bound algorithm which solves instances with up to 100 jobs and 3 common jump points optimally. We show in Section 1.4 that this special case in NP-hard in the ordinary sense. (Tseng et al., 2010) call the objective function *total stepwise tardiness*. The authors develop a Variable Neighborhood Search and apply it to $50-$job instances with 3, 4 and 5 jump points to obtain near-optimal feasible solutions. One of the contributions of this paper is a dedicated exact approach outperforming these computational results.

(Detienne et al., 2009) study a more general case on parallel unrelated machine with job release dates. The authors exhibit a set of dominant solutions for $R|r_j|\sum \bar{f}_j(C_j)$, and design an Integer Linear Programming (ILP) formulation based on this dominance. For the problem $R||\sum \bar{f}_j(C_j)$, they present another set of dominant solutions and an associated ILP formulation. Our current work relies on this formulation, which is recalled in Section 1.3. (Detienne et al., 2009) also evaluate the pertinence of the ILP models through the direct application of a Mixed Integer Programming solver.

The problem $1||\sum \bar{f}_j(C_j)$ is trivially a generalization of $1||\sum U_j$. The latter problem is polynomial and can be solved in $\mathcal{O}(n\log(n))$ (Moore, 1968). When jobs have different weights, the problem, denoted $1||\sum w_j U_j$, is NP-hard, even when due dates are equal (Karp, 1972). When release dates are introduced, the problem $1|r_j|\sum U_j$ is strongly NP-hard (Lenstra et al., 1977). (Baptiste et al., 2008) present an ILP formulation for the problem $1|\bar{d}_j|\sum w_j U_j$, in which each job must be scheduled before its deadline such that the weighted number of tardy jobs is minimized. Based on their formulation, the authors develop an exact method that can solve instances of $1|\bar{d}_j|\sum w_j U_j$ with up to 30000 jobs and instances of $1||\sum w_j U_j$ with up to 50000 jobs. Proposition 5 in Section 1.4 shows that $1|s_j = 2|\sum \bar{f}_j(C_j)$ (all jobs have two unequal jump points) is at least as hard as $1|\bar{d}_j|\sum w_j U_j$. Unfortunately, this paper is, to our knowledge, the latest work on this problem, and it is not clear for the authors whether this problem is NP-hard in the ordinary sense or strong sense.

Most of the work described in this paper relies on a Lagrangean relaxation of the number of occurrences of each job in an Integer Linear Programming formulation. This approach has been used several times in the literature for total cost scheduling problems, mainly starting from a time-indexed strategy, in various solving strategies. It is used in (Abdul-Razaq and Potts, 1988) and (Peridy et al., 2003) in a branch-and-bound approach. (Sourd, 2009) strengthens the relaxation by adding valid inequalities to develop a branch-and-bound approach. (Ibaraki and Nakamura, 1994) propose a Successive Sublimation Dynamic Programming algorithm. An obvious drawback of the relaxation is that there is no limitation to the number of times a job will appear in the pseudo-schedules representing optimal solutions of the Lagrangean subproblem, which can be composed of many occurrences of a single job and lead to poor lower bounds. In order to reduce this disadvantage and obtain a better bound, (Abdul-Razaq and Potts, 1988) and (Peridy et al., 2003) use the concept of short-term memory to prevent two occurrences of the same job to be processed consecutively in the subproblem, and (Peridy et al., 2003) extend the concept to any partial sequence of a fixed number of jobs. Their method is limited by the time and space complexity of the algorithm, which forbids its practical application to partial sequences of more than three jobs. The same concept is used by (Tanaka et al., 2009) to improve the approach of (Ibaraki and Nakamura, 1994), as well as other dominances on sequences of four jobs. Our approach differs from these previous works by using a strong formulation for our problem, in which the number of occurrences of each job in the solution of the Lagrangean subproblem is bounded by its number of jump points plus one, and such that they can appear only at specific positions. This yields very good lower bounds allowing the use of an alternative solving procedure described in Section 3. It consists of searching for a shortest feasible path in a graph embedding this dominance property, by successively removing infeasible shortest paths. We adapt and extend a basic filtering rule similar to the ones used in (Sourd, 2009) or (Tanaka et al., 2009) to reduce the size of the graph.

1.3. Integer Linear Programming models

This section provides two Binary Linear Programming models of the problem $1||\sum \bar{f}_j(C_j)$.

1.3.1. Time-indexed formulation

The problem can be written in the classical time-indexed formulation (Sousa and Wolsey, 1992). Note that this generic model cannot be used directly in practice because of the large number of variables it involves. Indeed, standard MILP solvers are not able to solve most programs corresponding to 30-job instances.

Let $P = \sum_{j \in J} p_j$ denote the total processing time of all jobs. Let us denote c_{jt} the cost of completing job $j \in J$ at time instant $t \in T$, with $T = \{0, \ldots, P\}$ the scheduling horizon. According to the previous notations, we have $c_{jt} = \gamma_j^k$, with k such that $d_j^{k-1} < t \leq d_j^k$. Let us introduce the set of decision variables $(x_{jt})_{j \in J, t \in T}$, such that x_{jt} is equal to 1 iff j completes at t, 0 otherwise. This allows us to write a time-indexed formulation of the problem:

$$
\begin{aligned}
(TI):\min \quad & \sum_{j \in J} \sum_{t=p_j}^{P} c_{jt} x_{jt} & & (1)\\
\text{s.t.} \quad & \sum_{t=p_j}^{P} x_{jt} = 1 & j \in J \quad & (2)\\
& \sum_{j \in J} \sum_{\theta=\max(t+1,p_j)}^{t+p_j} x_{j\theta} \leq 1 & t \in T \quad & (3)\\
& x_{jt} \in \{0,1\} & j \in J, t \in \{p_j, \ldots, P\} \quad & (4)
\end{aligned}
$$

In this model, Constraints (2) ensure that each job has exactly one completion time, and thus is processed exactly once, and Constraints (3) that no more than one job is executed at a time. Indeed, $\sum_{\theta=\max(t+1,p_j)}^{t+p_j} x_{j\theta}$ is equal to one if and only if job j is processed during time instant t.

1.3.2. Dominant order formulation

The model described in this section, which is the base of our approach, has been proposed by (Detienne et al., 2009) for the unrelated parallel machine variation of the problem. The ILP formulation relies on the following well-known result (see e.g. (Jackson, 1955)). Let us consider the problem of scheduling a set of jobs I with no release dates on a single processor and without preemption, such that each job $j \in I$ completes before its deadline $\bar{d}_j$.

Proposition 1. *If there exists at least one feasible schedule in which each job completes before its deadline, then the schedules where jobs are sequenced according to a non-decreasing order of their deadlines are feasible.*

This leads to the following characterization of feasible solutions.

Proposition 2. *Let σ be the permutation of I corresponding to a non-decreasing order of the deadlines. There exists at least one feasible solution iff:*

$$\sum_{r=1}^{j} p_{\sigma(r)} \leq \bar{d}_{\sigma(j)} \qquad j \in I$$

The problem $1||\sum_j \bar{f}_j(C_j)$ can clearly be seen as the problem of finding a minimum cost assignment of deadlines, corresponding to the jump points, where the cost functions change, such that there exists at least one feasible schedule meeting the deadlines. Using Proposition 2 for characterizing the set of feasible schedules leads to the following ILP model, based on the notion of *occurrences* of jobs. Each jump point d_j^l, $j \in J$, $l \in K_i$, is linked with a possible occurrence k of job j subject to a deadline d_k and whose execution cost is $\gamma_k = \gamma_j^l$, corresponding to the execution of j before $d_k = d_j^l$. Hence, η (which is equal to $\sum s_j$) is the total number of occurrences, and we assume that occurrences are indexed according to a non-decreasing order of their deadlines, i.e. $d_1 \leq d_2 \leq \cdots \leq d_\eta$. Let $\sigma(k)$, $k \in \{1, \ldots, \eta\}$ denote the job corresponding to the k^{th} occurrence. Finally, $\bar{K}_j = \{k \in \{1, \ldots, \eta\} | \sigma(k) = j\}$, $j \in J$, is the set of indices of the occurrences linked with job j. We can now rewrite the problem as:

$$(DOF) : \min \quad \sum_{k \in \{1,\ldots,\eta\}} \gamma_k \cdot u_k \tag{5}$$

$$\text{s.t.} \quad \sum_{k \in \bar{K}_j} u_k = 1 \qquad j \in J \tag{6}$$

$$\sum_{l=1}^{k} p_{\sigma(l)} \cdot u_l \leq d_k \qquad k \in \{1, \ldots, \eta\} \tag{7}$$

$$u_k \in \{0, 1\} \qquad k \in \{1, \ldots, \eta\} \tag{8}$$

In this model, decision variable u_k is equal to 1 if and only if occurrence k is processed. Constraints (6) ensure that exactly one occurrence of each job is processed, and Constraints (7) ensure that this selection of occurrences can lead to a feasible schedule, related to Proposition 2. This model can be seen as a generalization of a model used in the literature for the minimization of the weighted number of tardy jobs on a single machine (Lawler and Moore, 1969) or parallel machines (M'Hallah and Bulfin, 2005).

It is worth noting that ties on deadlines can be broken in an arbitrary way. In this case, Constraint (7) associated with the occurrence of highest rank trivially dominates the constraints associated with occurrences with the same deadline, which can thus be removed from the model. It follows that the number of Constraints (7) is equal to the number of distinct jump points. Let us recall that D denotes the set of distinct jump points. Hence, Constraints (7) can be replaced by:

$$\sum_{l\in\{1,\dots,\eta\},d_l\leq d} p_{\sigma(l)}\cdot u_l \leq d \quad d\in D \tag{9}$$

Besides, one can recognize a Multichoice Multidimensional Knapsack Problem in formulation (DOF): each group of items corresponds to a job, each item to an occurrence of job, and there is one resource constraint for each distinct deadline.

1.4. Complexity results

This section provides some complexity results on $1||\sum \bar{f}_j(C_j)$.

Proposition 3. *The problem $1||\sum \bar{f}_j(C_j)$ is NP-hard in the strong sense.*

Proof. Consider the special case of $1||\sum \bar{f}_j(C_j)$ in which each job j has a first jump point d_j^1 and $q_j = \sum_{i\in J} p_i - d_j^1$ other jump points such that $d_j^k = d_j^{k-1}+1$, $k = 2,\dots,q_j$ such that $\gamma_j^k = \gamma_j^{k-1}+v_j$, $k = 2,\dots,q_j$ and $v_j \in \mathbb{N}$. This instance defines an instance of $1||\sum w_jT_j$ with due dates $d_j = d_j^1$ and weights $w_j = v_j$, $j \in J$. Thus, the problem $1||\sum w_jT_j$, which is NP-hard in the strong sense (Lawler, 1977), is a special case of $1||\sum \bar{f}_j(C_j)$, which is also NP-hard in the strong sense. □

Proposition 4 focuses on the case where a fixed number $\alpha \in \mathbb{N}$ of distinct jump points is considered.

Proposition 4. *The problem $1\big||D| = \alpha\big|\sum \bar{f}_j(C_j)$ is NP-hard in the ordinary sense.*

Proof. The special case $\alpha = 1$ is a classical $0-1$-knapsack problem. Thus, $1\big||D| = \alpha\big|\sum \bar{f}_j(C_j)$ is NP-hard.

Moreover, the Multichoice Knapsack Problem can be solved by a Dynamic Programming procedure running in $\mathcal{O}(ab)$, with a the number of items involved and b the right-hand-side of the resource constraint (Dudzinski and Walukiewicz, 1987). A straightforward adaptation of this algorithm solves the Multichoice Multidimension Knapsack Problem in a $\mathcal{O}(aB^m)$, with B the maximum value of right-hand-sides of resource constraints, and m the number of resource constraints. The direct application of this procedure to the formulation (DOF) leads to an $\mathcal{O}(n\alpha P^\alpha)$ pseudo-polynomial algorithm to solve $1\big||D| = \alpha\big|\sum \bar{f}_j(C_j)$ optimally. Thus, $1\big||D| = \alpha\big|\sum \bar{f}_j(C_j)$ is NP-hard in the ordinary sense. □

Proposition 5. *The problem in which each job has two distinct jump points* $(1|s_j = 2|\sum \bar{f}_j(C_j))$ *is at least as hard as* $1|\bar{d}_j|\sum w_jU_j$.

Proof. The proof goes by showing that there is a polynomial transformation from $1|\bar{d}_j|\sum w_jU_j$ to $1|s_j = 2|\sum \bar{f}_j(C_j)$. Let Ω be an instance of the decision problem associated with $1|\bar{d}_j|\sum w_jU_j$, characterized by an integer Z, a set of jobs J and, for all $j \in J$, a processing time p_j, a deadline $\bar{d}_j$, a due date d_j and a weight w_j. The question is: Is there a feasible schedule with objective value less than or equal to Z?

Let us build an instance Ω' of the decision problem associated with $1|s_j = 2|\sum \bar{f}_j(C_j)$, composed of Z, a set of jobs J' with one job j' for each job $j \in J$, and, for each $j' \in J'$, a processing time $p_{j'} = p_j$ and two jump points:

- $d^1_{j'} = d_j$, whose associated cost is $\gamma^1_{j'} = w_j$,
- and $d^2_{j'} = \bar{d}_j$, whose associated cost is $\gamma^2_{j'} = \max(\sum_{i\in J} w_i, Z) + 1$.

Note that it is straightforward to derive a solution from Ω to Ω' by scheduling the corresponding jobs in the same order, and such that the completion times of the linked jobs are equal. Thus, if the answer to Ω is YES, it is easy to verify that we can derive a feasible schedule to Ω' from any feasible schedule of Ω, so that the answer to Ω' is YES. Suppose that the answer to Ω' is YES. Then, consider a feasible schedule of Ω'. For all $j' \in J'$, j' completes before or at time instant $d^2_{j'} = \bar{d}_j$ since $\gamma^2_{j'} > Z$. So, one can derive a feasible schedule for Ω that has trivially the same cost, and the answer to Ω is YES. □

2. Lagrangean bounds

This section describes a lower bound and an upper bound, both based on a Lagrangean relaxation.

2.1. Lower bound by Lagrangean relaxation

This section presents a lower bound derived from a Lagrangean relaxation of the (DOF) model. Let us consider the model made of Constraints (5), (6), (9) and (8), and let us add the following valid equality in order to strengthen the relaxation:

$$\sum_{k=1}^{\eta} p_{\sigma(k)} \cdot u_k = P \tag{10}$$

This simply specifies that the total processing time in the schedule must be equal to the total processing time of the jobs. Let $\pi = (\pi_1, \dots, \pi_n)$ denote Lagrangean multipliers associated with the n Constraints (6), and let us price out these constraints to get the dual function below:

$$\begin{aligned} L(\pi) = \min \quad & \sum_{k=1}^{\eta} (\gamma_k + \pi_{\sigma(k)}) \cdot u_k - \sum_{j \in J} \pi_j && (11) \\ \text{s.t.} \quad & (9) \text{ and } (8) \text{ and } (10) && (12) \end{aligned}$$

It follows that the dual problem $[DP]$ consists in finding a vector of Lagrangean multipliers π maximizing $L(\pi)$:

$$[DP] = \max_{\pi \in \mathbb{R}^n} L(\pi)$$

The resulting value provides a lower bound for (DOF). The Lagrangean sub-problem can be interpreted as the problem of finding a timed sequence of correctly ordered occurrences respecting the disjunctive constraint, whose total duration is P and of minimum Lagrangean cost.

Let $\tilde{\gamma}_k = \gamma_k + \pi_{\sigma(k)}$ denote the Lagrangean cost of the variable u_k. This subproblem can be solved using the following Dynamic Programming equations:

$$\left\{ \begin{array}{lll} f(0,0) = & 0 & \\ f(k,t) = & \min \left\{ \begin{array}{l} f(k-1,t) \\ f(k-1, t - p_{\sigma(k)}) + \tilde{\gamma}_k \end{array} \right. & k \in \{1, \dots, \eta\}, t \in \{0, \dots, d_k\} \\ f(k,t) = & \infty & \text{otherwise.} \end{array} \right.$$

In these recurrence equations, $f(k, t), k \in \{1, \ldots, \eta\}, t \in \mathbb{N}$, denotes the minimum possible Lagrangean cost of a sequence composed of a subset of the k first occurrences in the EDD order, of length t. The first relation states initial conditions: If no occurrence is processed at the beginning of the horizon, the cost is null. The second equation decomposes into two alternatives:

- Either occurrence k is processed and completes at time t, and the partial optimal cost is equal to the optimal cost of a schedule composed of a subset of the first $k-1$ occurrences completing just before k starts, plus the cost of processing k,
- Or k is not processed or does not complete at time t, and the partial optimal cost is equal to the optimal cost of a schedule completing at t and composed of a subset of the first $k-1$ occurrences.

The last relation corresponds to infeasible states. The optimal value is obtained by calculating $f(\eta, P)$, which can be done using a straightforward $\mathcal{O}(\eta P)$-time procedure. The Lagrangean lower bound can be maximized using a sub-gradient procedure (see e.g. Held et al., 1974).

2.2. Upper bound by Lagrangean heuristic

The heuristic provides an upper bound of the optimum of our problem, by deriving a feasible solution from a solution of the Lagrangean subproblem. The idea is, first, to randomly (uniformly) select one occurrence per job scheduled in the Lagrangean solution. Then, the jobs that are not in the Lagrangean solution are inserted, one by one, in a greedy way, at the position leading to the partial schedule of smallest cost. The insertion order is randomly determined.

This procedure takes advantage of the fact that the best insertion position of a job into a sequence of q jobs can be found in $\mathcal{O}(q)$-time, although an isolated evaluation of the cost of a solution can only be done in $\mathcal{O}(q)$-time. Indeed, as there are no availability constraints on the machine or jobs and the cost function is regular, solutions without idle times are dominant. Focusing on this set of solutions, the cost difference incurred when interchanging two adjacent jobs $j1$ and $j2$, $j1$ initially starting at t_0, is given by $c_{j2,t_0+p_{j2}} + c_{j1,t_0+p_{j2}+p_{j1}} - (c_{j1,t_0+p_{j1}} + c_{j2,t_0+p_{j1}+p_{j2}})$. Thus, one can first append the job to be inserted to the sequence and evaluate the cost of the new sequence in $\mathcal{O}(1)$-time complexity. Then, the job is interchanged with the preceding job in the sequence and the cost of the new sequence is incrementally evaluated in $\mathcal{O}(1)$-time complexity. The operation is repeated until the beginning of the sequence, leading to an overall complexity of $\mathcal{O}(q)$. The counterpart for this time complexity is the necessity to have a $\mathcal{O}(1)$-time access to the cost c_{jt} of completing a job j at time instant t, which requires an $\mathcal{O}(nP)$-space storage. As the procedure is very fast, it can be used many times, providing different schedules.

The best sequence obtained is then improved using a simple local search algorithm. The neighborhood explored consists in extracting and re-inserting one job at another position in the sequence. Note that scanning the whole neighborhood is made by a $\mathcal{O}(n^2)$-time procedure, by applying the same trick as in the heuristic in order to reduce the complexity of the search.

3. Exact method

As stated in the literature review (Section 1.2), several papers deal with a graph representation of scheduling problems and filtering rules on this graph. To our knowledge, the best solving approaches are branch-and-bound methods (e.g., Sourd, 2009) or Successive Sublimation Dynamic Programming (Ibaraki and Nakamura, 1994; Tanaka et al., 2009).

As shown in the numerical experimental of Section 4.3, the quality of the Lagrangean bounds led us to develop an exact method. The idea is to correct the nearly feasible Lagrangean solution to obtain an optimal solution. The principle is, thus, to work with the graph representation of the Dynamic Program presented in Section 2.1, and to look for a feasible shortest path in this graph – corresponding to a feasible schedule – by successively removing infeasible shortest paths, i.e. infeasible schedules.

3.1. Occurrence/Time-indexed Graph

Let us consider an instance of $1||\bar{f}(C_j)$ and its ordered set of job occurrences, and let us build the associated *Occurrence/Time-indexed Graph* $G = (X, E, c, \rho, \omega)$, where:

- X is the set of vertices,
- E is the set of edges,
- c_e is the cost associated to edge e,

- ρ_e is the subset of occurrences associated to edge e,
- ω_e is the subset of jobs associated to edge e.

Moreover, given any vector of Lagrangean multipliers π, we define $\tilde{c}_e \in \mathbb{R}$ as the Lagrangean cost associated edge e.

The idea is to map the set of possible pseudo-schedules into the set of paths in the graph G, whose costs are equal to the costs of the pseudo-schedules, and to map the set of schedules into the set of feasible paths in G subject to additional constraints on the jobs associated to edges. Let us formally define the structure of G. It is a layered graph in which each node $x_{k,t}$ represents a decision point, i.e. the possibility to start or skip occurrence k at time t. Each occurrence is associated with the nodes of one layer, and an additional fictive node x_* represents the end of the schedule:

$$\begin{aligned} X &= \left(\bigcup_{k \in \{1,\dots,\eta\}} X_k \right) \cup \{x_*\}, \\ X_k &= \left\{ x_{k,t} | t \in \{0,\dots,d_k\} \right\}, \qquad\qquad k \in \{1,\dots,\eta\}. \end{aligned}$$

An edge $e = (x_{k,t}, x_{k',t'})$ represents the processing of ρ_e, a subset of the occurrences between occurrences k (included) and k' (not included), at starting time t, such that the machine is available for the occurrence k' at time t'. Recall (see Section 1.3.2) that $\sigma(k)$ denotes the job corresponding to the k^{th} occurrence, with $k \in \{1,\dots,\eta\}$ and where $\eta = \sum s_j$ is the total number of occurrences.

Basically, if we consider a singleton $\{k\}$ of occurrences, the first alternative is that k is not processed at t so that the machine is available for processing $k+1$ at time instant t:

$$E_0 = \left\{ \left(x_{k,t}, x_{k+1,t}\right) | k \in \{1,\dots,\eta-1\}, t \in \{0,\dots,d_k\} \right\} \cup \left\{ (x_{\eta,P}, x_*) \right\}.$$

We have, for $e \in E_0$, $\rho_e = \emptyset$ and the cost for these edges is null ($c_e = 0$, $\tilde{c}_e = 0$). The second alternative is that k is processed at t so that the machine is not available for occurrence $k+1$ until time instant $t + p_{\sigma(k)}$:

$$\begin{aligned} E_1 = \Bigg\{ & \left(x_{k,t}, x_{k+1,t+p_{\sigma(k)}}\right) | k \in \{1,\dots,\eta-1\}, \\ & t \in \left\{0,\dots,\max\left(d_k, d_{k+1} - p_{\sigma(k)}\right)\right\} \Bigg\} \cup \left\{ (x_{\eta,P-p_{\sigma(\eta)}}, x_*) \right\}. \end{aligned}$$

In this case, we have, for $e = (x_{k,t}, x_{k',t'}) \in E_1$, $c_e = \gamma_k$, $\tilde{c}_e = \tilde{\gamma}_k$ and $\rho_e = \{k\}$. Finally, we have $E = E_0 \cup E_1$ and $\omega_e = \{\sigma(k) | k \in \rho_e\}, e \in E$.

This basic graph is meant to be reduced by the rules described below, in order to improve the efficiency of computing optimal pseudo-schedules and schedules. Let us introduce the following definitions, which are used in the sequel.

Definition 1. *An admissible edge $e \in E$ is an edge that includes at most one occurrence of each job, i.e. $\sigma(k) \neq \sigma(l), k \in \rho_e, l \in \rho_e, k \neq l$.*

Any feasible schedule corresponds to a path that contains only admissible edges, and non-admissible edges can be removed from the graph.

Definition 2. *An admissible path $(e_1, \ldots, e_r)$ of edges e_i in G is a path such that:*

- *$e_1 = (x_{1,0}, x_{k,t})$, i.e. the path starts from the node corresponding to the first occurrence at time instant* 0,
- *$e_r = (x_{k',t'}, x_*)$, i.e. the path ends at the fictive node representing the end of the schedule,*
- *$\omega_{e_i} \cap \omega_{e_l} = \emptyset$, $i \in \{1, \ldots, r\}$, $l \in \{1, \ldots, r\}$, $i \neq l$ (i.e. there is no job that appears twice in the corresponding pseudo-schedule),*
- *$\cup_{l \in \{1,\ldots,r\}} \omega_{e_l} = J$ (i.e. each job is in the corresponding pseudo-schedule).*

3.2. Non Lagrangean relaxation based rules for reducing the size of the graph

These rules are meant to be applied on the graph G described above, as well as on an already modified graph obtained from G. For the sake of conciseness, we do not give formal proofs for the time complexity of the filtering procedures.

A non-ramified path $(e_1, \ldots, e_r)$ can trivially be replaced by a single edge ϵ, such that $\omega_\epsilon = \bigcup_{l=1}^{r} \omega_{e_l}$, $c_\epsilon = \sum_{l=1}^{r} c_{e_l}$ and $\tilde{c}_\epsilon = \sum_{l=1}^{r} \tilde{c}_{e_l}$. Note that a path $(e_1, \ldots, e_r)$ is called non-ramified if e_{i-1} and e_{i+1} are the only edges connected to edge e_i, $\forall i = 2, \ldots, r-1$. Merging non-ramified paths allows subsequent operations on G to be performed faster, since the number of edges is reduced. Besides, a non-ramified path that contains edges including more than once a given job can be removed, since it would correspond to a non-admissible edge. From an implementation point of view, merging all non-ramified paths takes

$\mathcal{O}(n|E|)$ operations, for scanning all edges and checking the admissibility of a new edge.

The first rule benefits from the fact that all edges corresponding to a subset of occurrences associated with one job may have been removed from the graph. As any admissible path must contain at least one edge of the remaining subset of edges including a job j, any edge starting "before" all remaining occurrences and ending "after" them, while not including j, cannot be part of an admissible path and can be removed from the graph. Let us introduce the following notations, used in Propositions 6 and 7. Let $j \in J$, and define $O(j) = \{k \in \{1, \dots, \eta\} \mid \exists\ x_{k',t'} \in X, ((x_{k,t}, x_{k',t'}) \in E) \ \wedge\ (j \in \omega_{(x_{k,t}, x_{k',t'})})\}$, the set of occurrences still possible for j, $o^-(j) = \arg\min\{k \in O(j)\}$ and $o^+(j) = \arg\max\{k \in O(j)\}$ the occurrences still possible for j of lower rank and higher rank respectively.

Proposition 6. *Let $e = (x_{k,t}, x_{k',t'}) \in E$. If for some job $j \notin \omega_e$, $k \leq o^-(j)$, and $k' > o^+(j)$, then e cannot be part of an admissible path.*

Proof. Straightforward: In any path passing through e, j cannot be before e, after e, or by e. $\square$

The rules below apply on the parts of the graph where a path passing through a given node necessarily passes through a given arc.

Proposition 7. *Let $k' \in \{1, \dots, \eta\}$ such that, for some job j, $k' > o^+(j)$, and $x_{k',t'} \in X_{k'}$ such that $\{(x_{k',t'}, x_{k'',t''}) \in E\} = \{e\}$, i.e. only one edge e starts from $x_{k',t'}$. Any edge $f = (x_{k,t}, x_{k',t'}) \in E$ such that $j \notin \omega_e \cup \omega_f$, and $k \leq o^-(j)$ cannot be part of an admissible path.*

Proof. Any path passing through f also passes through e. Thus, in such a path, j cannot be before e, by e, by f or after f. $\square$

Proposition 8. *Let $x_{k,t} \in X$ such that $\{(x_{k,t}, x_{k',t'}) \in E\} = \{e\}$, i.e. only one edge e starts from node $x_{k,t}$. Any edge $f = (x_{k'',t''}, x_{k,t}) \in E$ ending at node $x_{k,t}$ such that $\omega_e \cap \omega_f \neq \emptyset$ cannot be part of an admissible path.*

Proof. Straightforward: Any path passing through f also passes through e, and includes at least one job more than once. $\square$

The preceding propositions can also be written for the symmetric case, i.e. when the inner degree of a given node is equal to one, and can be implemented in $\mathcal{O}(n|E|)$ time.

The following rule simply compares the length of longest possible paths passing through a given edge with a known lower bound of the length of an optimal admissible path. If the longest path is shorter than the lower bound, the node (resp. arc) cannot be part of an admissible shortest path, and can be removed from the graph.

Proposition 9. *For all $x_{k,t} \in X$, let $l^-_{max}(x_{k,t})$ (resp. $l^+_{max}(x_{k,t})$) be the length of a longest path from node $x_{1,0}$ to node $x_{k,t}$ (resp. from node $x_{k,t}$ to node x_*), and LB a lower bound of the length of a shortest admissible path. Let $e = (x_{k,t}, x_{k',t'}) \in E$.*
If $l^-_{max}(x_{k,t}) + \gamma_e + l^+_{max}(x_{k',t'}) < LB$, then e cannot be part of a shortest admissible path.

Proof. $l^-_{max}(x_{k,t}) + \gamma_e + l^+_{max}(x_{k',t'})$ is the length of a longest path passing through e. Since there is no path passing through e and of length larger than LB, and the length of any admissible path is larger than or equal to LB, no shortest admissible paths pass through e. □

Computing all the values $l^-_{max}(x_{k,t})$, $x_{k,t} \in X$, can be performed using Bellman's shortest path algorithm, in $\mathcal{O}(|E|)$ time. The values $l^+_{max}(x_{k,t})$ can be computed using a symmetric method. Finally, scanning the edges takes $\mathcal{O}(|E|)$ operations, leading to an overall time complexity of $\mathcal{O}(|E|)$ for detecting removable edges using this proposition.

3.3. Lagrangean relaxation based rules to reduce the size of the graph

These rules are based on the Lagrangean bound presented in Section 2.1. Let us recall the expression of the Lagrangean sub-problem:

$$L(\pi) = \min \quad \sum_{k=1}^{\eta} (\gamma_k + \pi_{j(k)}) \cdot u_k - \sum_{j \in J} \pi_j$$
$$\text{s.t.} \qquad (9) \text{ and } (8) \text{ and } (10).$$

As $\sum_{j \in J} \pi_j$ is constant relatively to variables $(u_k)_{k \in K}$, computing $L(\pi)$ consists in finding a shortest path in the layered graph G from $x_{1,0}$ to x_*, applying costs $\tilde{c}$ on the arcs. Formally, we can write:

$$L(\pi) = d^-(x_*) - \sum_{j \in J} \pi_j \qquad \forall \pi \in \mathbb{R}^n,$$

where $d^-(x_{k,t}), x_{k,t} \in X$, denotes the cost of a shortest path in G from $x_{1,0}$ to $x_{k,t}$, relatively to the Lagrangean costs $\tilde{c}$.

In the sequel, a "shortest path" in G refers to a shortest path relative to the Lagrangian costs $\tilde{c}$, and we assume that a vector π of Lagrangean multipliers is known, as well as an upper bound UB of the optimum of the problem. Proposition 10 is similar to the filtering rules used in (Sourd, 2009) and (Tanaka et al., 2009) in our specific graph.

Proposition 10. *Let us define the following notations:*

- $d^-(x_{k,t}), x_{k,t} \in X$, *is the cost of a shortest path in* G *from* $x_{1,0}$ *to* $x_{k,t}$, *relatively to the Lagrangian costs* $\tilde{c}$,
- $d^+(x_{k,t}), x_{k,t} \in X$, *is the cost of a shortest path in* G *from* $x_{k,t}$ *to* x_*, *relatively to the Lagrangian costs* $\tilde{c}$,
- $e = (x_{k,t}, x_{k',t'}) \in E$ *is a given edge in* G.

If $d^-(x_{k,t}) + \tilde{c}_e + d^+(x_{k',t'}) > UB + \sum_{j \in J} \pi_j$, *then* e *cannot be part of a shortest admissible path.*

Proof. $d^-(x_{k,t}) + \tilde{c}_e + d^+(x_{k',t'})$ is the cost of a shortest path passing through e, and $UB + \sum_{j \in J} \pi_j$ is an upper bound of the cost of a shortest admissible path in G. Hence, the result holds. □

This rule can be applied to remove unnecessary arcs, using an $\mathcal{O}(|E|)$ algorithm. Indeed, computing $d^-(x_{k,t}), x_{k,t} \in X$, (resp. $d^+(x_{k,t}), x_{k,t} \in X$) takes $\mathcal{O}(|E|)$ operations using Bellman's algorithm. Once these values are known, scanning every edge in the graph takes $\mathcal{O}(|E|)$ operations. This proposition can be extended to the propositions below.

Proposition 11. *Let us define the following notations:*

- $j \in J$ *is a given job,*
- $d^-_{\neg j}(x_{k,t}), x_{k,t} \in X$, *is the cost of a shortest path* $(e_1, \ldots, e_r)$ *in* G *from* $x_{1,0}$ *to* $x_{k,t}$, *relatively to the Lagrangian costs* $\tilde{c}$, *and which does not include job* j $(\{e \in (e_1, \ldots, e_r)$ *s.t.* $j \in \omega_e\} = \emptyset)$,
- $d^+_{\neg j}(x_{k,t}), x_{k,t} \in X$, *is the cost of a shortest path* $(e_1, \ldots, e_r)$ *in* G *from* $x_{k,t}$ *to* x_*, *relatively to the Lagrangian costs* $\tilde{c}$, *and which does not include job* j $(\{e \in (e_1, \ldots, e_r)$ *s.t.* $j \in \omega_e\} = \emptyset)$,

- *$e = (x_{k,t}, x_{k',t'}) \in E$ is a given edge in G, which includes job j ($j \in \omega_e$).*

If $d^-_{\neg j}(x_{k,t}) + \tilde{c}_e + d^+\neg j(x_{k',t'}) > UB + \sum_{j \in J} \pi_j$, then e cannot be part of a shortest admissible path.

Proof. Any admissible path must pass exactly once through an edge including j. So, any admissible path passing through e has to be composed only of edges which do not include j. $d^-_{\neg j}(x_{k,t}) + \tilde{c}_e + d^+_{\neg j}(x_{k',t'})$ is the minimum cost of such an admissible path. As $UB + \sum_{j \in J} \pi_j$ is an upper bound of the cost of a shortest admissible path, the result holds. □

Proposition 12. *Let us define the following notations:*

- *$j \in J$ is a given job,*
- *$d^-_j(x_{k,t})$, $x_{k,t} \in X$, is the cost of a shortest path $(e_1, \dots, e_r)$ in G from $x_{1,0}$ to $x_{k,t}$, relatively to the Lagrangian costs $\tilde{c}$, and which includes job j exactly once ($|\{e \in (e_1, \dots, e_r)$ s.t. $j \in \omega_e\}| = 1$),*
- *$d^+_j(x_{k,t})$, $x_{k,t} \in X$, is the cost of a shortest path $(e_1, \dots, e_r)$ in G from $x_{k,t}$ to x_*, relatively to the Lagrangian costs $\tilde{c}$, and which includes job j exactly once ($|\{e \in (e_1, \dots, e_r)$ s.t. $j \in \omega_e\}| = 1$),*
- *$e = (x_{k,t}, x_{k',t'}) \in E$ is a given edge in G, which does not include job j ($j \notin \omega_e$),*
- *UB is a known upper bound of $Opt(QTS_1)$,*
- *$\pi \in \mathbb{R}^n$ is a vector of Lagrangian multipliers.*

If $\min(d^-_{\neg j}(x_{k,t}) + \tilde{c}_e + d^+_j(x_{k',t'}), d^-_j(x_{k,t}) + \tilde{c}_e + d^+_{\neg j}(x_{k',t'})) > UB + \sum_{j \in J} \pi_j$, then e cannot be part of a shortest admissible path.

Proof. Any admissible path must pass exactly once through an edge including j. So, for any admissible path passing through e, two cases can occur:

- j is *before* e, and $d^-_j(x_{k,t}) + \tilde{c}_e + d^+_{\neg j}(x_{k',t'})$ is a lower bound of the cost of such an admissible path,
- Or j is *after* e, and $d^-_{\neg j}(x_{k,t}) + \tilde{c}_e + d^+_j(x_{k',t'})$ is a lower bound of the cost of such an admissible path.

As $UB + \sum_{j \in J} \pi_j$ is an upper bound of the cost of a shortest admissible path, the result holds. $\square$

In order to apply Propositions 11 and 12, one needs to determine the values $d_j^-(x_{k,t})$, $d_j^+(x_{k,t})$, $d_{\neg j}^-(x_{k,t})$ and $d_{\neg j}^+(x_{k,t})$, for $j \in J$ and $x_{k,t} \in X$. For a given job j, this can be achieved in two passes (forward and backward) of a labeling procedure which applies the following recurrence equations:

$$\left\{ \begin{array}{rl}
d_{\neg j}^-(x_{1,0}) = & 0 \\
d_{\neg j}^-(x_{k,t}) = & \min_{(x_{k',t'},x_{k,t}) \in E, j \notin \omega_{(x_{k',t'},x_{k,t})}} (d_{\neg j}^-(x_{k',t'}) + \tilde{c}_{x_{k',t'},x_{k,t}}) \\
& \qquad k \in \{1,\dots,\eta\}, t \in \{0,\dots,d_k\} \\
d_j^-(x_{1,0}) = & \infty \\
d_j^-(x_{k,t}) = & \min \left\{ \begin{array}{l} \min_{(x_{k',t'},x_{k,t}) \in E, j \notin \omega_{(x_{k',t'},x_{k,t})}} (d_j^-(x_{k',t'}) + \tilde{c}_{x_{k',t'},x_{k,t}}) \\ \min_{(x_{k',t'},x_{k,t}) \in E, j \in \omega_{(x_{k',t'},x_{k,t})}} (d_{\neg j}^-(x_{k',t'}) + \tilde{c}_{x_{k',t'},x_{k,t}}) \end{array} \right. \\
& \qquad k \in \{1,\dots,\eta\}, t \in \{0,\dots,d_k\} \\
d_{\neg j}^+(x_*) = & 0 \\
d_{\neg j}^+(x_{k,t}) = & \min_{(x_{k,t},x_{k',t'}) \in E, j \notin \omega_{(x_{k,t},x_{k',t'})}} (d_{\neg j}^+(x_{k',t'}) + \tilde{c}_{x_{k,t},x_{k',t'}}) \\
& \qquad k \in \{1,\dots,\eta\}, t \in \{0,\dots,d_k\} \\
d_j^+(x_*) = & \infty \\
d_j^+(x_{k,t}) = & \min \left\{ \begin{array}{l} \min_{(x_{k,t},x_{k',t'}) \in E, j \notin \omega_{(x_{k,t},x_{k',t'})}} (d_j^+(x_{k',t'}) + \tilde{c}_{x_{k,t},x_{k',t'}})\} \\ \min_{(x_{k,t},x_{k',t'}) \in E, j \in \omega_{(x_{k,t},x_{k',t'})}} (d_{\neg j}^+(x_{k',t'}) + \tilde{c}_{x_{k,t},x_{k',t'}}) \end{array} \right. \\
& \qquad k \in \{1,\dots,\eta\}, t \in \{0,\dots,d_k\}
\end{array} \right.$$

Thus, computing all the shortest distances takes $\mathcal{O}(n|E|))$ operations. Scanning the edges takes $\mathcal{O}(n|E|))$ additional operations, which leads to an overall time complexity of $\mathcal{O}(n|E|))$ for applying Propositions 11 and 12. One can note that Propositions 11 and 12 used together dominate Propositions 6, 7 and 8: For any node $x_{k,t}$ for which one of these propositions is active, either Proposition 11 or Proposition 12 allows all in-going or out-going edges of $x_{k,t}$ to be removed since the corresponding shortest distance is infinite. Moreover, this dominance is strict. The dominated propositions are still useful because their lower complexity allows them to be used more frequently in our method. The idea in Propositions 11 and 12 can be extended to deal with pairs of jobs. However, preliminary experiments indicate that the computational time required by this procedure (time complexity of $\mathcal{O}(n^2|E|))$ makes it difficult to use.

Algorithm 1 Main exact solving procedure

```
1: Compute initial Lagrangean multipliers $\pi$ and an upper bound
2: Build the initial graph $G_0$ ; $i \leftarrow 0$
3: repeat
4:    Update $\pi$ and perform graph filtering on $G_i \to G_{i+1}$
5:    $i \leftarrow i + 1$
6:    repeat
7:       Find a shortest path $\mu_i$ in $G_i$ and compute a heuristic solution
8:       if $\mu_i$ is not feasible then
9:          Remove $\mu_i$ from $G_i \to G_{i+1}$
10:         $i \leftarrow i + 1$
11:      end if
12:   until $\mu_i$ is feasible or $MaxIter$ iterations have been performed
13: until $\mu_i$ is feasible
```

Algorithm 1 summarizes the main procedure of our exact method. It consists in successively filtering the graph and removing infeasible paths, until an optimal feasible path is found. Filtering allows reducing the computing time of subsequent operations and reducing the number of paths to remove until finding a feasible one. Removing infeasible paths also allows the lower bound to be improved each time a shortest path is calculated and, thus, makes filtering rules more efficient.

The Lagrangean multipliers π and an upper bound are first initialized using the Lagrangean bounding procedure described in Section 2. From π, a set of useless dynamic programming states is determined using a simple adaptation of Proposition 10. This phase allows building an already reduced graph, and is necessary to limit memory consumption on large instances. An iteration of the procedure begins by updating π using a modified sub-gradient procedure *(Line 4)*. As in the initializing phase, an upper bound is derived at each iteration, and possibly improved. Moreover, Proposition 10 is applied at each iteration.

Every 30 iterations of the sub-gradient procedure, Propositions 6, 7 and 8 are also applied, while Propositions 11 and 12 are used every 60 iterations. Although both are very effective, their computing times prevent a more frequent use. Proposition 9 is triggered each time the best lower bound exceeds a new integer value. In our implementation, preliminary numerical studies led us to empirically choose $MaxIter = 200$.

Then, the procedure proceeds with the path removing phase *(Line 7)*. At each step, a path from node $x_{1,0}$ to x_* is calculated using Bellman's algorithm. Its feasibility is then checked. If it is feasible, then the current path corresponds to an optimal schedule. Indeed, since the relaxed constraints are all equality constraints, the Lagrangean and normal costs of the path are, in this case, equal. It follows that this value is both an upper bound and a lower bound of the optimum value of the objective function.

If the current path is not feasible, we first determine a maximal set of disjoint infeasible sub-paths, as described in Algorithm 2.

The procedure first determines a set of infeasible sub-paths (*Lines* 2–11), by identifying all pairs of edges of the path representing a same job. Each pair of edges clearly delimits an infeasible sub-path. By construction, the set of sub-paths is restricted to the ones containing at most two occurrences of the same job. We then proceed to the selection of a maximal subset of disjoint sub-paths using a dynamic programming algorithm (*Lines* 12-20). Note that this algorithm can be easily adapted to deal with weights on the sub-paths. However, we empirically chose not to weight sub-paths. Finally, the edges delimiting the sub-paths are identified using the classical backward procedure of a dynamic programming algorithm (*Lines* 21-26).

Each selected infeasible sub-path is then removed from the graph as described in Algorithm 3. This procedure roughly consists in removing the first edge of the sub-path (*Line* 14). In order to keep the possibility of alternate paths going through this edge but not through the whole sub-path being removed, each other sub-path in the graph which initially starts from the edge is replaced by one edge (*Lines* $3-13$). The paths that are replaced are part of the sub-graph composed of the levels only crossed by the sub-path being removed. Moreover, the admissibility of each created edge is checked (*Line* 6).

4. Numerical results

4.1. Instance generation

All methods presented in this paper are evaluated on a test bed composed of randomly generated instances. We focus on instances with only a small number of jump points per job. This choice is motivated by the fact that the more jump points, the more the problem is close to the general total

Algorithm 2 Identification of infeasible disjoint sub-paths

```
1: Let $(x_{k1,t1}, \dots, x_{kr,tr}, x_{kr+1,tr+1})$ be an infeasible path, with $x_{k1,t1} = x_{1,0}$
   and $x_{kr+1,tr+1} = x_*$
2: $\forall j \in J$, $lastProcessedOccurrence[j] \leftarrow \emptyset$
3: $I \leftarrow \emptyset$
4: for $z$ from 1 to $r$ do
5:     for all $j \in \omega_{(x_{kz,tz}, x_{kz+1,tz+1})}$ do
6:         if $lastProcessedOccurrence[j] \neq \emptyset$ then
7:             $I \leftarrow I \cup (lastProcessedOccurrence[j], kz)$
8:         end if
9:         $lastProcessedOccurrence[j] \leftarrow kz$
10:    end for
11: end for
12: $\forall z \in \{1, \dots, r\}$, $F[z] \leftarrow 1$ ; $pred[z] \leftarrow \emptyset$
13: for $z$ from $r - 1$ to 1 do
14:     Let $(i_z^-, i_z^+) \leftarrow I_z$
15:     for $y$ from $z + 1$ to $r$ do
16:         if $F[z] < F[y] + 1$ and $i_y^- > i_z^+$ then
17:             $F[z] \leftarrow F[y] + 1$ ; $pred[z] \leftarrow y$
18:         end if
19:     end for
20: end for
21: $z^* = \arg\max\{F[z], z \in \{1, \dots, r\}\}$
22: $optimalSubSet \leftarrow \emptyset$
23: while $z^* \neq \emptyset$ do
24:     $optimalSubSet \leftarrow optimalSubSet \cup I_{z^*}$
25:     $z^* \leftarrow pred[z^*]$
26: end while
```

Algorithm 3 Removing a sub-path from the graph

```
1: Let $(x_{k1,t1}, \dots, x_{kr,tr})$ denote the sub-path to be removed
2: $J' \leftarrow \emptyset$
3: for $z$ from 2 to $r$ do
4:     $J' \leftarrow J' \cup \omega_{(x_{kz-1,tz-1}, x_{kz,tz})}$
5:     for all $(x_{kz,tz}, x_{k',t'}) \in E$ do
6:         if $J' \cap \omega_{(x_{kz,tz}, x_{k',t'})} = \emptyset$ then
7:             $E \leftarrow E \cup (x_{k1,t1}, x_{k',t'})$
8:             with $\omega_{(x_{k1,t1}, x_{k',t'})} = J' \cup \omega_{(x_{kz,tz}, x_{k',t'})}$,
9:                 $\rho_{(x_{k1,t1}, x_{k',t'})} = (\bigcup_{y=1}^{z-1} \rho_{(x_{ky,ty}, x_{ky+1,ty+1})}) \cup \rho_{(x_{kz,tz}, x_{k',t'})}$
10:                $c_{(x_{k1,t1}, x_{k',t'})} = (\sum_{y=1}^{z-1} c_{(x_{ky,ty}, x_{ky+1,ty+1})}) + c_{(x_{kz,tz}, x_{k',t'})}$
11:        end if
12:    end for
13: end for
14: $E \leftarrow E - (x_{k1,t1}, x_{k2,t2})$
```

cost scheduling problem, and the stepwise structure vanishes. Our generator takes as inputs n, the number of jobs, and K, the number of jump points per job. For each job j, a processing time p_j is drawn from a uniform distribution $\{1, \dots, 100\}$, and K jump points are drawn from a uniform distribution $\{p_j, \dots, \sum_i p_i\}$. For each jump point of each job, a cost increase is drawn from a uniform distribution $\{1, \dots, 100\}$. For each combination of the parameters $n \in \{10, 20, 30, 50, 100, 200, 300, 400, 500\}$ and $K \in \{2, 3, 4, 9\}$, 10 instances are generated, leading to a total of 360 instances. All tests are performed on a laptop PC with a 2.5 GHz Intel Core2-Duo processor and 3.5 GB RAM, running Windows XP Pro.

4.2. Lagrangian relaxation

The results reported in this section are obtained by running the subgradient procedure described in Section 2.1. At each iteration of the algorithm, except in the 100 first iterations to save computing time, the heuristic is ran a given number of times which is reported in the tables below as the parameter *runs*. The best solution of these runs is compared to the best upper bound obtained by the heuristic so far. When it is better, the local search method is used to improve the solution.

Tables 1 and 2 compare the quality and computing time of our Lagrangean lower bound ((DOF) *Lag.*) with the linear relaxations of the (DOF) ((DOF)

	(DOF) Lag.		(DOF) Linear		(TI) Linear	
n	Gap	Time (sec)	Gap	Time (sec)	Gap	Time (sec)
10	0.07%	0.0	34.51%	0.0	12.91%	0.2
20	0.31%	0.2	14.58%	0.0	7.29%	2.9
30	0.34%	0.5	10.80%	0.0	6.09%	9.4
50	0.27%	2.1	6.04%	0.0	3.82%	77.9
100	0.17%	9.4	2.84%	0.1	*	*
200	0.11%	43.4	1.50%	0.4	*	*
300	0.15%	98.3	0.95%	1.0	*	*
400	0.18%	183.2	0.81%	2.0	*	*
500	0.22%	288.7	0.72%	3.3	*	*
Mean	0.20%	69.9	8.08%	0.8	7.53%	22.6

* The solver went out of memory for all these instances.

Table 1: Computing times and average gaps between lower bounds and best known upper bounds, according to parameter n.

	(DOF) Lag.		(DOF) Linear		(TI) Linear	
K	Gap	Time (sec)	Gap	Time (sec)	Gap	Time (sec)
2	0.04%	30.9	9.72%	20.1	9.01%$^+$	0.1$^+$
3	0.15%	53.3	8.90%	23.9	8.44%$^+$	0.2$^+$
4	0.26%	68.1	7.84%	22.7	8.58%$^+$	0.4$^+$
9	0.36%	127.3	3.65%	23.8	6.30%$^+$	2.4$^+$
Mean	0.20%	69.9	7.53%	22.6	8.08%$^+$	0.8$^+$

$^+$ The solver went out of memory for instances with more than 50 jobs.

Table 2: Computing times and average gaps between lower bounds and best known upper bounds, according to parameter K.

Linear) and the (TI) ((TI) *linear*) formulations. The gap value reported for a relaxation value V is equal to $(Best \quad upper \quad bound - V)/V$. The upper bound used is the optimal value if it is known, or the best upper bound obtained by any method used in the paper. The quality of the Lagrangean bound is excellent and dominates the bound obtained by the time-indexed formulation, which is known to provide good bounds for total cost problems but is limited by memory requirements. As expected, the computing time of (DOF) *Lag.* becomes prohibitive when the number of jobs is too large. In our experiments, the number of iterations of the sub-gradient procedure never exceeds 600. Although the computing time of the linear relaxation

	runs					
n	2	5	10	50	100	200
10	0.07%	0.07%	0.07%	0.07%	0.07%	0.07%
20	0.46%	0.49%	0.44%	0.43%	0.39%	0.37%
30	0.65%	0.50%	0.49%	0.45%	0.44%	0.44%
50	0.70%	0.59%	0.49%	0.45%	0.42%	0.42%
100	0.63%	0.59%	0.50%	0.39%	0.39%	0.33%
200	0.60%	0.48%	0.43%	0.32%	0.28%	0.24%
300	0.65%	0.54%	0.48%	0.38%	0.34%	0.32%
400	0.55%	0.44%	0.41%	0.32%	0.30%	0.27%
500	0.54%	0.47%	0.42%	0.35%	0.31%	0.29%
Mean Heur.	0.54%	0.46%	0.42%	0.35%	0.33%	0.30%
Mean Heur+LS	0.46%	0.41%	0.37%	0.31%	0.29%	0.27%

Table 3: Gaps between Lagrangean upper and lower bounds, according to parameter n.

of (DOF) remains very small even for the largest instances, its relative gap decreases when the number of jobs increases. The same phenomenon applies for (TI) *Linear*, and also when the number of jump points increases. It can be explained by the fact that the value of the objective function increases with these parameters, and that the gap induced by the continuous relaxation becomes relatively smaller (but still large in absolute values). It is interesting to note that this does not apply to our Lagrangean relaxation, which keeps a large part of the problem integer.

Table 3 reports gaps between the Lagrangean heuristic and the Lagrangean relaxation. The gaps are very small and the use of the local search allows reducing them even more (cf. Line *Mean Heur.+LS*). Preliminary results indicate that the results do not improve significantly over 200 runs of the heuristic at each iteration of the sub-gradient. The total computing time required by the heuristic during the sub-gradient procedure is reported in Table 4. The total computing time of the local search procedure for 500-job instances is 0.8 seconds on average.

4.3. Exact method

This section analyses the number of instances optimally solved by two approaches: A mathematical programming solver and the Infeasible Path Removing approach using the set of propositions of Section 3.

	$runs$					
n	2	5	10	50	100	200
10	0.0	0.0	0.0	0.0	0.0	0.0
20	0.0	0.0	0.0	0.1	0.2	0.3
30	0.0	0.0	0.0	0.1	0.3	0.5
50	0.0	0.0	0.1	0.4	0.7	1.3
100	0.0	0.1	0.2	1.1	2.3	4.2
200	0.2	0.4	0.9	3.8	8.2	14.7
300	0.4	0.9	1.6	7.7	16.5	29.8
400	0.7	1.5	3.0	13.3	28.8	51.1
500	1.1	2.4	4.4	23.1	45.2	81.9
Mean	0.3	0.6	1.1	5.5	11.4	20.4

Table 4: Total computing time (sec.) required by the Lagrangean heuristic, according to parameter n.

	n									
K	10	20	30	50	100	200	300	400	500	Total
2	10	10	10	10	10	10	10	10	10	90/90
3	10	10	10	10	10	10	10	9	8	87/90
4	10	10	10	10	10	10	8	0	0	68/90
9	10	10	10	10	6	0	0	0	0	46/90
Total	40/40	40/40	40/40	40/40	36/40	30/40	28/40	19/40	18/40	291/360

Table 5: Number of instances solved optimally by XPRESS-MP on model (DOF) in 3600 seconds, according to parameters n and K.

	n									
K	10	20	30	50	100	200	300	400	500	Mean
2	0,0	0,0	0,0	0,0	0,4	2,0	4,5	12,6	21,2	4,5
3	0,0	0,0	0,1	0,5	2,4	31,2	425,5	538,4	1393,0	225,8
4	0,0	0,1	0,4	1,0	15,5	278,2	1265,3			195,1
9	0,1	0,7	7,2	33,2	1220,4					168,1
Mean	0,0	0,2	1,9	8,7	208,5	103,8	515,1	261,7	586,1	140,3

Table 6: Average computing time required by XPRESS-MP on formulation (DOF), according to parameters n and K. Values are calculated on instances solved by both methods.

	n									
K	10	20	30	50	100	200	300	400	500	Total
2	10	10	10	10	10	10	10	10	10	90/90
3	10	10	10	10	10	10	10	**10**	**10**	**90**/90
4	10	10	10	10	10	10	**10**	**9**	**8**	**87**/90
9	10	10	10	10	**10**	**8**	**3**	0	0	**61**/90
Total	40/40	40/40	40/40	40/40	**40**/40	**38**/40	**33**/40	**29**/40	**28**/40	**328**/360

Table 7: Number of instances solved optimally by the Infeasible Path Removing approach in 3600 seconds, according to parameters n and K.

	n									
K	10	20	30	50	100	200	300	400	500	Mean
2	0,0	0,0	0,1	0,4	2,6	18,3	40,9	80,1	146,3	32,1
3	0,0	0,1	0,2	1,3	5,9	34,6	125,9	183,2	572,0	91,9
4	0,0	0,2	0,5	2,4	10,7	39,8	306,9			44,7
9	0,0	0,6	1,5	4,8	28,4					5,2
Mean	0,0	0,2	0,6	2,2	10,1	30,9	147,3	128,9	335,5	48,5

Table 8: Average computing time required by the Infeasible Path Removing approach, according to parameters n and K. Values are calculated on instances solved by both methods.

	n									
K	10	20	30	50	100	200	300	400	500	Total
2	10	10	10	9	3	0	0	0	0	42/90
3	10	10	10	10	1	0	0	0	0	41/90
4	10	10	10	9	1	0	0	0	0	40/90
9	10	10	10	10	4	0	0	0	0	44/90
Total	40/40	40/40	40/40	38/40	9/40	0/40	0/40	0/40	0/40	167/360

Table 9: Number of instances solved optimally by the SSDP method of (Tanaka et al., 2009) in 3600 seconds, according to parameters n and K.

Tables 5, 6, 7 and 8 report the mathematical programming results with a one-hour time limit, as well as the average computing time required for instances solved optimally by both methods. In these experiments, the number of runs of the Lagrangean heuristic is empirically fixed to 50. These tables show that our method can globally be favorably compared to the direct application of the solver on the most appropriate (to our knowledge) ILP formulation for the problem. However, it can be noticed that XPRESS-MP performs well on instances with two jump points per job ($K = 2$). Our method also outperforms the approaches proposed in (Tseng et al., 2010) and (Yang, 2009).

We also report in Table 9 the results obtained with a one-hour time limit with the SSDP method of (Tanaka et al., 2009), which is not dedicated to our problem. The results are quite poor since no problem with 200 jobs are solved optimally. We believe this is due to the high quality of our lower bound, that comes up with solutions that are very close to be feasible.

A naive branch-and-bound method exploiting these bounds has also been tested. It solves all instances with up to 300 jobs and 2 jump points, 100 jobs and 3 or 4 jump points, and 50 jobs and 9 jump points within a one-hour time limit. For the sake of conciseness, we do not detail this method or numerical results, and just report its performance for comparison.

In order to evaluate the pertinence of the different rules described in Section 3, Tables 10 and 11 report the number of instances optimally solved relatively to the set of Propositions implemented in the Infeasible Path Removing method, while Tables 12 and 13 give the number of paths removed from the graph before the procedure finds the optimal feasible path. The adapted classical rule alone already gives better results than the mathematical programming solver. Introducing the non Lagrangean filtering rules reduces the average number of iterations of the procedure and allows more instances to be solved. The best results are obtained when using Propositions 11 and 12, which divide the number of paths to remove by up to 5 on average for instances with 9 jump points.

5. Conclusion

A Lagrangean relaxation based bounding scheme is presented for the problem $1||\sum \bar{f}_i(C_i)$. The very good bounds obtained are used in a solving procedure based on a specific dominance property of the problem and extensions of existing Lagrangean filtering rules. Numerical experiments show that this approach can be favorably compared with a naive branch-and-bound method, the direct use of a mathematical programming solver and with previously published approaches. A straightforward adaptation of the method has been tested for the problem $1|r_i|\sum \bar{f}_i(C_i)$, based on the model described by (Detienne et al., 2009). The method does not perform so well in the presence of release dates, since it cannot solve some 30 and 50-job instances: It appears that the lower bound is sometimes hardly improved by the path removing procedure. Further work on the approach could consist in designing an adapted graph to this generalization or stronger bounds.

More generally, we believe that the study of scheduling problems with step cost functions as objective is an interesting research topic, because these functions bring a gradation to the binary late/on-time concept that is not taken into account by classical objectives as, for example, total tardiness, which supposes a linear increase of the cost.

Abdul-Razaq, T. S., Potts, C. N., 1988. Dynamic programming State-Space relaxation for Single-Machine scheduling. The Journal of the Operational Research Society 39 (2), 141–152.

	n									
Rules	10	20	30	50	100	200	300	400	500	Total
Prop. 10	40	40	40	40	40	36	29	26	23	314
Prop. 10, 6, 7, 8, 9	40	40	40	40	40	37	30	28	26	321
Prop. 10, 6, 7, 8, 9, 11, 12	40	40	40	40	40	38	33	29	28	328

Table 10: Number of instances optimally solved within 3600 seconds, according to filtering rules and parameter n.

	K				
Rules	2	3	4	9	Total
Prop. 10	90	87	80	57	314
Prop. 10, 6, 7, 8, 9	90	89	84	58	321
Prop. 10, 6, 7, 8, 9, 11, 12	90	90	87	61	328

Table 11: Number of instances optimally solved within 3600 seconds, according to filtering rules and parameter K.

	K				
Rules	10	20	30	50	100
Prop. 10	0,2	9,9	35,5	78,8	445,5
Prop. 10, 6, 7, 8, 9	0,0	4,0	26,6	64,8	338,3
Prop. 10, 6, 7, 8, 9, 11, 12	0,0	2,5	21,6	49,8	163,5

	K				
Rules	200	300	400	500	Total
Prop. 10	2491,1	2016,8	1915,6	1621,7	821,9
Prop. 10, 6, 7, 8, 9	1518,1	1538,9	1613,8	1390,5	606,9
Prop. 10, 6, 7, 8, 9, 11, 12	461,4	567,8	649,1	748,2	244,1

Table 12: Average number of paths removed from the graph before finding the optimal feasible path, according to parameter n. Values are computed on instances solved with the three configurations only.

	n				
Rules	2	3	4	9	Total
Prop. 10	55,3	585,3	1263,3	1773,8	821,9
Prop. 10, 6, 7, 8, 9	46,6	462,1	1031,0	1117,4	606,9
Prop. 10, 6, 7, 8, 9, 11, 12	28,9	214,9	449,7	340,1	244,1

Table 13: Average number of paths removed from the graph before finding the optimal feasible path, according to parameter K. Values are computed on instances solved with the three configurations only.

Baptiste, P., Croce, F. D., Grosso, A., T'kindt, V., 2008. Sequencing a single machine with due dates and deadlines: an ILP-based approach to solve very large instances. Journal of Scheduling 13 (1), 39–47.

Brucker, P., 2004. Scheduling Algorithms. SpringerVerlag.

Curry, J., Peters, B., 2005. Rescheduling parallel machines with stepwise increasing tardiness and machine assignment stability objectives. International Journal of Production Research 43, 3231–3246.

Detienne, B., Dauzère-Pérès, S., Yugma, C., 2009. Scheduling inspection operations subject to a fixed production schedule. In: Proceedings of the 4th Multidisciplinary International Scheduling Conference MISTA 2009, Dublin, Ireland.

Dudzinski, K., Walukiewicz, S., 1987. Exact methods for the knapsack problem and its generalizations. European Journal of Operational Research 28 (1), 3 – 21.

Graham, R. L., Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 4, 287–326.

Held, M., Wolfe, P., Crowder, H., 1974. Validation of subgradient optimization. Mathematical Programming 6, 62–88.

Ibaraki, T., Nakamura, Y., 1994. A dynamic programming method for single machine scheduling. European Journal of Operational Research 76 (1), 72–82.

Jackson, J. R., 1955. Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, University of California.

Karp, R. M., 1972. Reducibility among combinatorial problems. In: Miller, R. E., Thatcher, J. W. (Eds.), Complexity of Computer Computations. Plenum Press, pp. 85–103.

Lawler, E. L., 1977. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathematics Volume 1, 331–342.

Lawler, E. L., Moore, J. M., 1969. A functional equation and its application to resource allocation and sequencing problems. Management Science 16 (1), 77–84.

Lenstra, J., Kan, A. R., Brucker, P., 1977. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1 (1), 343–362.

M'Hallah, R., Bulfin, R. L., 2005. Minimizing the weighted number of tardy jobs on parallel processors. European Journal of Operational Research 160 (2), 471 – 484, decision Support Systems in the Internet Age.

Moore, J. M., 1968. A n job one machine algorithm for minimizing the number of late jobs. Management Science 15, 102–109.

Peridy, L., Pinson, E., Rivreau, D., 2003. Using short-term memory to minimize the weighted number of late jobs on a single machine. European Journal of Operational Research 148 (0), 591–603.

Sahin, G., 2006. New combinatorial approaches for solving railroad planning and scheduling problems. Ph.D. thesis, University of Florida, USA.

Sourd, F., 2009. New exact algorithms for one-machine earliness-tardiness scheduling. INFORMS J. on Computing 21 (1), 167–175.

Sousa, J. P., Wolsey, L. A., 1992. A time indexed formulation of non-preemptive single machine scheduling problems. Mathematical Programming 54 (1), 353–367.

Tanaka, S., Fujikuma, S., Araki, M., 2009. An exact algorithm for single-machine scheduling without machine idle time. Journal of Scheduling 12 (6), 575–593.

Tseng, C.-T., Chou, Y.-C., Chen, W.-Y., October 2010. A variable neighborhood search for the single machine total stepwise tardiness problem. In: Proceedings of the 2010 International Conference on Engineering, Project and Production Management. Pingtun, Taiwan, pp. 101–108.

Yang, H., 2009. Maximizing total profit of jobs with stepwise non-increasing profit under multiple common due dates. In: Proceedings of the 20th Annual POMS conference. Orlando, Florida, pp. 1041–1050.