Alcom-FT Technical Report Series
ALCOMFT-TR-01-106

Some comments on sequencing with controllable processing times

Han Hoogeveen *   Gerhard J. Woeginger †

Abstract

We discuss sequencing problems on a single machine with controllable job processing times. For the maximum job cost criterion, we present several polynomial time results. For the total weighted job completion time criterion, we present an NP-hardness result. Our results settle several open questions in this area.

Keywords. Scheduling, sequencing, contollable processing time, single machine, computational complexity, bicriterion optimization.

1 Introduction

Most classical scheduling models assume *fixed* job processing times, i.e., the processing time of every job is determined a priori and cannot be modified by the scheduler. However, in real-life applications the processing of a job often requires additional resources such as facilities, manpower, funds, fuel, catalyzer, raw materials, and so on. By increasing or decreasing these additional resources, shorter job processing times may be accomplished. As a consequence compressing a job is possible at some extra cost. A real-life example of such a problem is discussed by Janiak [5] in the context of steel production.

In the theory of scheduling, these situations are usually modelled as follows (cf. e.g. Nowicki & Zdrzałka [8]): There are n jobs $J_1, \ldots, J_n$. The processing requirement of a job J_j is specified by three non-negative parameters a_j, u_j, c_j where $0 \le u_j \le a_j$. Here a_j is the initial processing requirement of J_j, u_j is a bound on the compressibility of J_j, and c_j models the compression cost of J_j. By assigning additional resources to the processing of job J_j, the *actual* processing time of job J_j may be compressed down to $p_j = a_j - x_j$ where the variable x_j with $0 \le x_j \le u_j$ models the amount of compression. The cost of performing this compression equals $c_j x_j$. Solving a scheduling problem with controllable processing times now amounts to (i) specifying a feasible schedule π for the jobs together with (ii) specifying a compression vector $\vec{x}$ that encodes the compressions x_j of all the jobs. For an arbitrary optimality criterion f (like the makespan $C_{\max}$ or the maximum lateness $L_{\max}$), we denote

*E-mail: `slam@cs.uu.nl`. Department of Computer Science, Utrecht University, P.O.Box 80089, 3508 TB Utrecht, The Netherlands. Supported by EC Contract IST-1999-14186 (Project alcom-FT).

†Email: `gwoegi@opt.math.tu-graz.ac.at`. Institut für Mathematik B, TU Graz, Steyrergasse 30, A-8010 Graz, Austria, and Department of Mathematics, University of Twente, 7500 AE Enschede, The Netherlands. Supported by the START program Y43-MAT of the Austrian Ministry of Science.

by $F_1(f, \pi, \vec{x})$ the cost of the schedule π under criterion f with processing times compressed by $\vec{x}$. Moreover, we denote by

$$F_2(\vec{x}) \doteq \sum_{j=1}^{n} c_j x_j \tag{1}$$

the total compression cost of $\vec{x}$. Hence, scheduling with controllable job processing times essentially is a *bicriteria* problem, and the following four basic optimization problems arise.

P1. Minimize $F_1(f, \pi, \vec{x}) + F_2(\vec{x})$.

P2. Minimize $F_2(\vec{x})$ under the constraint $F_1(f, \pi, \vec{x}) \leq \tau$.

P3. Minimize $F_1(f, \pi, \vec{x})$ under the constraint $F_2(\vec{x}) \leq \kappa$.

P4. Identify the trade-off curve for (F_1, F_2).

The trade-off curve connects all points (F_1, F_2) where F_1 is the best possible value of $F_1(f, \pi, \vec{x})$ given that $F_2(\vec{x}) \leq F_2$, and vice versa. Note that an efficient solution to P4 also solves the problems P1–P3 as a by-product. We denote scheduling with controllable job processing times by an entry '*contr*' in the second field of the standard three-field scheduling notation (see e.g. Graham, Lawler, Lenstra & Rinnooy Kan [4] and Lawler, Lenstra, Rinnooy Kan & Shmoys [7]).

Let us briefly recollect some results from the literature on scheduling with controllable job processing times. Vickson [12, 13] studies P1 for problems $1|contr|\sum C_j$ and $1|contr|\sum w_j C_j$. For $1|contr|\sum C_j$, problem P1 can be formulated as an assignment problem and hence can be solved in $O(n^{2.5})$ time. For problem $1|contr|\sum C_j$, Ruiz Diaz & French [10] develop an enumerative algorithm for P4. They also note that the trade-off curve in general is not convex. Vickson [13] describes an enumerative algorithm for P4 for $1|contr|\sum w_j C_j$. Alidaee & Ahmadian [1] observe that the results of Vickson [12] can be extended to parallel machines, which yields a polynomial time algorithm for P1 for the problem $P|contr|\sum C_j$. Nowicki & Zdrzałka [9] give an $O(n^2)$ greedy algorithm to solve P4 for problem $P|contr|C_{\max}$; in this case, the trade-off curve is piecewise linear with up to $2n+1$ breakpoints. Van Wassenhove & Baker [14] show that problems P1–P4 are solvable in polynomial time for $1|contr|f_{\max}$ under the assumption that $f_j(t) = w_j t$ for all $j = 1, \ldots, n$. Janiak & Kovalyov [6] analyze problem P2 for $1|contr|f_{\max}$ and show that it is solvable in $O(n \log n)$ time if the x_j values are allowed to be fractional and that it is NP-hard in the ordinary sense, otherwise. Cheng, Chen, and Li [2] provide a pseudo-polynomial algorithm to solve P4 for $1|contr|\sum U_j$ and show that this is best possible by establishing ordinary NP-hardness of this problem; the entry $\sum U_j$ indicates that the number of jobs completing after their due date is to be minimized.

Contribution and organization of this paper. In Section 2 we deal with the problem of minimizing the maximum job cost with controllable job processing times. We show that for piecewise linear penalty functions, all four problems P1–P4 are polynomially solvable by extending the approach by Van Wassenhove & Baker [14]. This solves a problem that has been open since the paper by Van Wassenhove & Baker [14]. In Section 3 we deal with the problem of minimizing the total weighted job completion time with controllable job processing times. We prove that for this objective function, problem P1 is NP-hard. This solves a problem that has been open since the papers by Vickson [12, 13].

2 Minimizing the maximum job cost

In this section we discuss the single machine scheduling problem of minimizing the maximum cost with controllable job processing times. For every job J_j ($j = 1, \ldots, n$), there is a corresponding non-decreasing penalty function $f_j : \mathbb{R}_0^+ \to \mathbb{R}_0^+$. All jobs are available for processing at time 0. If a job J_j completes at time C_j in some schedule, then the cost incurred by J_j equals $f_j(C_j)$. The goal is to minimize the maximum of these job costs. This leads to the scheduling problem $1|contr|f_{\max}$ with controllable processing times where

$$F_1(f, \pi, \vec{x}) \;=\; \max_{1 \le j \le n} \; f_{\pi(j)} \left(\sum_{i=1}^{j} (a_{\pi(i)} - x_{\pi(i)}) \right), \tag{2}$$

and where $F_2(\vec{x})$ is as specified in (1). Van Wassenhove & Baker [14] investigate the special case of this problem where the penalty functions f_j are totally ordered, i.e., where

$$f_j(t) \;\le\; f_{j+1}(t) \qquad \text{for all } t \text{ and all } 1 \le j \le n-1. \tag{3}$$

For the case of totally ordered *linear* penalty functions f_j, Van Wassenhove & Baker [14] describe polynomial time $O(n^3)$ solutions for all four problems P1–P4 for $1|contr|f_{\max}$. Tuzikov [11] presents a polynomial time algorithm for problem P2 for $1|contr|f_{\max}$ with arbitrary non-decreasing penalty functions f_j. The computational complexity of the problems P1, P3, and P4 for $1|contr|f_{\max}$ with arbitrary non-decreasing penalty functions f_j is unknown.

In this section, we consider the special case of problem $1|contr|f_{\max}$ where all penalty functions f_j are piecewise linear. Not surprisingly, we must carefully consider the *representation* of these penalty functions in the input. It is instructive to start with the following example for the case where the penalty functions f_j are specified by an oracle algorithm.

Example 2.1 *(A bad example for oracle representations)*
Consider an instance I of the NP-hard satisfiability problem (Garey & Johnson [3]): Instance I consists of a Boolean formula Φ over k Boolean variables, and the question is whether there exists a satisfying truth-assignment for Φ. With any integer i, $0 \le i \le 2^k - 1$, we associate the truth-setting for the k Boolean variables that results from interpreting the k bits in the binary representation of i as truth-values for the k Boolean variables.

Consider the instance I' of $1|contr|f_{\max}$ that consists of a single job J_1 with parameters $a_1 = 2^k - 1$, $u_1 = 2^k - 1$, and $c_1 = 1$, and with the following piecewise linear and non-decreasing penalty function $f_1 : \mathbb{R}_0^+ \to \mathbb{R}_0^+$.

$$f_1(t) \;=\; \begin{cases} \lceil t \rceil & \textit{if } t \le 2^k - 1 \textit{ and if } \lceil t \rceil \textit{ encodes a satisfying truth-setting} \\ \lceil t \rceil + 1 & \textit{otherwise} \end{cases}$$

It can be verified that the optimal solution to problem P1 for instance I' always will use an integral compression x_1 of job J_1. For such an integral compression, the overall cost in problem P1 equals $f_1(2^k - 1 - x_1) + x_1$. Therefore, the optimal solution to problem P1 for instance I' has value 2^k if formula Φ is unsatisfiable, and value $2^k - 1$ if formula Φ is satisfiable.

As a consequence, an algorithm for the above instance I' of $1|contr|f_{\max}$ can be used to separate satisfiable from unsatisfiable formulas (and that is an NP-complete problem). Does this mean that the scheduling problem $1|contr|f_{\max}$ with piecewise linear penalty functions f_j itself is an NP-hard problem? Well, the answer depends on the representation of the penalty functions f_j in the input. In the above example, the piecewise linear penalty function f_1 is represented *implicitly* via the Boolean formula Φ: For any given real t, we can round up t to the next integer $\lceil t \rceil$, translate $\lceil t \rceil$ into binary representation, and check in polynomial time whether the resulting bits satisfy the formula Φ. Hence, the penalty function f_1 is specified via a so-called *polynomial time oracle algorithm.* And for this (admittedly somewhat strange) representation of the input, solving problem $1|contr|f_{\max}$ to optimality indeed is NP-hard.

Observation 2.2 *Solving problem P1 for $1|contr|f_{\max}$ is NP-hard, even if the penalty functions f_j all are piecewise linear and non-decreasing functions that are specified by a polynomial time oracle algorithm.* ■

We remark that it is straightforward to modify the construction in Example 2.1 such that the penalty function f_1 additionally is continuous. Given the result in Observation 2.2, we turn to a more natural, *explicit* representation of the piecewise linear penalty functions f_j as specified in the following definition.

Definition 2.3 (*Explicitly represented piecewise linear penalty functions*)
For $j = 1, \ldots, n$, the penalty function f_j is a piecewise linear and non-decreasing function. It is specified by

- *a positive integer m_j that specifies the number of break points,*
- *a sequence $0 = z_j^0 < z_j^1 < \ldots < z_j^{m_j} = \infty$ of break points,*
- *a sequence of non-decreasing linear functions $f_j^i : \mathbb{R}^+ \to \mathbb{R}^+$, $i = 1, \ldots, m_j$.*

For $i = 1, \ldots, m_j$, the function f_j equals f_j^i on the interval $(z_j^{i-1}, z_j^i]$. Moreover, $f_j(0) = 0$. By $L = \sum_{j=1}^n m_j$ we denote the total number of linear pieces of all functions f_j. ■

Now let us introduce three sets S_1, S_2, S_3 of geometric points.

- The graph of the linear piece induced by the function f_j^i is a straight line segment that connects a so-called left geometric point with coordinates $x = z_j^{i-1}$ and $y = f_j^i(z_j^{i-1})$ to a so-called right geometric point with coordinates $x = z_j^i$ and $y = f_j^i(z_j^i)$. Since any such linear piece is left-open and right-closed, the left point is not part of the graph, whereas the right point is part of the graph. We denote the set of all these left and right end-points of all these linear pieces by S_1. Note that $|S_1| = O(L)$ and that S_1 can be computed in polynomial time.
- Next, observe that these L linear pieces intersect in at most $O(L^2)$ points. We denote the set of all these intersection points by S_2. Note that $|S_2| = O(L^2)$ and that S_2 can be computed in polynomial time.
- Finally, consider the intersection points of the functions f_j with the vertical line $x = \sum_{j=1}^n a_j$. Denote the set of these $O(n)$ intersection points by S_3. Also S_3 can be determined in polynomial time.

Let $y_1 < y_2 < \ldots < y_\ell$ denote the sorted sequence of y-coordinates of the points in $S_1 \cup S_2 \cup S_3$. Note that $\ell = O(L^2 + n)$. Define $y_0 = 0$ and $y_{\ell+1} = \infty$. We break down the set of feasible solutions into $\ell + 1$ solution classes: The kth class ($k = 1, \ldots, \ell + 1$) contains those feasible solutions whose objective value $F_1(f, \pi, \vec{x})$ is contained in the interval $(y_{k-1}, y_k]$.

We take a closer look at the kth solution class. Consider the behavior of the graph of function f_j between the two horizontal lines $y = y_{k-1}$ and $y = y_k$. This graph either is a line segment that connects these two lines, or it stays below the line $y = y_{k-1}$ completely; if the function jumps from a value less or equal to y_{k-1} to a value greater or equal to y_k, then we depict this by a vertical line at the corresponding time point. Observe that none of the line segments intersect in this horizontal stripe, except for possibly in the endpoint. Moreover, if f_j stays below y_{k-1}, then clearly it will have cost at most y_{k-1} when it is completed at the end of the schedule, independent of the compression vector $\vec{x}$; hence, we can ignore such a job when looking for a schedule with cost $F_1(f, \pi, \vec{x}) \in (y_{k-1}, y_k]$. Therefore, we assume without loss of generality that the graph of function f_j is present between the horizontal lines $y = y_{k-1}$ and $y = y_k$. We renumber the jobs according to the position of their line segment in the stripe, where the job with the leftmost line segment gets number 1.

Observation 2.4 *Let π be any schedule with $y_{k-1} \leq F_1(f, \pi, \vec{x}) \leq y_k$. Reordering the jobs in π in order of increasing indices without changing the amount of compression per job does not increase the cost.*

Proof. A simple job interchange argument. Consider two consecutive jobs J_i and J_j with $i > j$ in an optimal solution from this class. Then swapping the two jobs (without changing their compression) does not increase the value of F. By repeating such swaps, we will eventually reach a solution of the desired form. ■

Observation 2.4 enables us to solve problem P4 for each solution class $(y_{k-1}, y_k]$ $(k = 1, \ldots, \ell + 1)$ through the algorithm designed by Van Wassenhove and Baker to solve problem P4 in case of totally ordered linear penalty functions f_j. They use the total ordering of the f_j's to show that in any schedule of interest the order of the jobs is always the same; this matches Observation 2.4. Then, given a corner point of the efficient frontier, they determine the F_1-value of the next corner point of the efficient frontier and the corresponding F_2-value by minimizing $F_2(\vec{x})$ subject to the constraint that $F_1(f, \pi, \vec{x}) \leq F_1$, etc.

In our case, we start with $F_1 = y_k$. The constraint $F_1 \leq y_k$ induces a deadline $\bar{d}_j$ for each job j, where $\bar{d}_1 \leq \bar{d}_2 \leq \ldots \leq \bar{d}_n$, because of our numbering. The problem of finding the corresponding F_2-value boils down to finding the compression vector $\vec{x}$ with minimum $\sum c_j x_j$ value such that each job is completed by its deadline. Van Wassenhove and Baker have shown that this problem is solvable through a greedy algorithm, which runs in $O(n^2)$ time; Janiak and Kovalyov proposed a similar algorithm that runs in $O(n \log n)$ time. Below, we propose another algorithm to solve this problem. This algorithm is based on solving a special case of a min-cost max-flow problem, which can be done in $O(n)$ time once the compression costs have been sorted. A further advantage of using this algorithm is that the number of iterations needed to compute the efficient frontier can be bounded more easily.

Our algorithm consists of two parts. In the first part, the algorithm computes the total compression b_j of the jobs $1, \ldots, j$ needed to have job j completed by its deadline; obviously, we have that

$$b_j \leftarrow \max\{0, \sum_{i=1}^{j} a_i - \bar{d}_j\}.$$

Next, the algorithm solves the problem of finding the values x_j that minimize

$$\begin{array}{lll} \min & \sum_{j=1}^{n} c_j x_j & \\ \text{s.t.} & 0 \leq x_j \leq u_j & \text{for } j = 1, \ldots, n \\ & \sum_{i=1}^{j} x_i \geq b_j & \text{for } j = 1, \ldots, n \end{array}$$

Since all $x_j \geq 0$, we assume without loss of generality that $b_j \geq b_{j-1}$ for $j = 1, \ldots, n$, where $b_0 = 0$. Since all $c_j > 0$, we know that the optimum solution satisfies the equation $\sum_{j=1}^{n} x_j = b_n$.

We formulate this linear programming problem as a min-cost max-flow problem as follows. We introduce a digraph $G = (V, A)$, where V contains the vertices $1, \ldots, n$, which correspond to the jobs, and two dummy vertices: source s and sink t. We introduce arcs (s, j) $(j = 1, \ldots, n)$ with capacity $b_j - b_{j-1}$ and cost 0; arcs $(j+1, j)$ $(j = 1, \ldots, n-1)$ with capacity ∞ and cost 0; and arcs (j, t) $(j = 1, \ldots, n)$ with capacity u_j and cost c_j. In this network we are looking for the cheapest way to send an amount of b_n units of flow from s to t.

Theorem 2.5 *The solution of the min-cost max-flow problem corresponds to an optimum solution of the LP-problem.*

Proof. Suppose that we have found a solution to the min-cost max-flow problem; denote the flow through the arc (j, t) by x_j $(j = 1, \ldots, n)$. We will show that this set of x_j values corresponds to an optimum solution of the LP-problem. To this end, we are done if we can show that each solution to the LP-problem corresponds to a flow of size b_n in the network and vice-versa, since the cost functions of both problems are equal.

We start with the constraints $0 \leq x_j \leq u_j$. Obviously, the flow through the arc (j, t) must satisfy this constraint, because of the capacity of (j, t). Conversely, if x_j is such that it satisfies $0 \leq x_j \leq u_j$, then a flow of this size fits through arc (j, t).

Now the second set of constraints. Note that a flow of size b_n will saturate all arcs (s, j) $(j = 1, \ldots, n)$. Hence, the total flow through the arcs $(s, 1), \ldots, (s, j)$ is equal to b_j, and this amount of flow must be drained off through the arcs $(1, t), \ldots, (j, t)$, which implies that $\sum_{i=1}^{j} x_i \geq b_j$ for all $j = 1, \ldots, n$. Note that the excess is exactly equal to the flow through the arc $(j+1, j)$. Hence, any flow of size b_n corresponds to a feasible solution of the LP-problem. The converse follows immediately. ∎

We solve the min-cost max-flow problem by iteratively pumping flow through the cheapest augmenting path. Any path from s to t has the form $(s, k)(k \to j), (j, t)$, where $(k \to j)$ denotes the path from k to j consisting of the arcs $(k, k-1), \ldots, (j+1, j)$; obviously, we need $k \geq j$. The cost of this path is equal to c_j, which implies that we should look for the cheapest unsaturated arc (j, t) and the smallest $k \geq j$ such that (s, k) can contain more flow; in case of a tie in cost, we choose the path in which the index j is minimal. If all arcs (s, k) with $k \geq j$ are saturated, then we cannot use the arc (j, t) anymore. Since in each iteration we saturate at least one of the $2n$ arcs (s, j) and (j, t), this algorithm can be implemented to run in $O(n)$ time after the c_j values have been sorted.

Solving the problem of minimizing $F_2(\vec{x})$ subject to the constraint that $F_1(f, \pi, \vec{x}) \leq y_k$ yields the first cornerpoint of the efficient frontier. We continue in the same fashion as Van Wassenhove and Baker. We decrease the upper bound y_k on $F_1(f, \pi, \vec{x})$ by an amount Δ, which results in a decrease of the deadlines $\bar{d}_j$ by $v_j \Delta$, where v_j is the reciprocal of the gradient of the cost function f_j in the interval $(z_j^{i-1}, z_j^i]$ that we are currently dealing with; we define $v_0 = 0$. The problem of finding the cheapest way of compressing the jobs $1, \ldots, n$ such that these deadlines are met can then again be formulated as a linear programming problem, where for a given value of Δ the second set of constraints then becomes

$$\sum_{i=1}^{j} x_i \geq b_j + v_j \Delta;$$

this can be solved as a min-cost max-flow problem as described above for any value of Δ.

To find the trade-off curve, we need to find the values of Δ that correspond to cornerpoints of the efficient frontier. It is easily verified that the slope of the efficient frontier will change only if one of the following events occurs:

1. some job j gets either fully compressed or fully decompressed;

2. the capacity of some arc (s, j) either drops to zero or emerges from zero.

Given a cornerpoint of the efficient frontier, it is straightforward to compute the value of Δ at which the first such event occurs; the only question is the number of these events.

We will first show that we can bound the number of times that job j can get fully compressed or decompressed by $O(n)$ between two successive events of type 2. In our analysis, we will assume without loss of generality that $b_{j+1} + v_{j+1}\Delta > b_j + v_j\Delta$, for all $j = 1, \ldots, n-1$, that is, each arc (s, j) has a positive capacity. After that, we will show that at most $O(n)$ times an event of type 2 can occur, which bounds the total number of cornerpoints of the efficient frontier to $O(n^3)$ for each solution class $(y_{k-1}, y_k]$ $(k = 1, \ldots, \ell+1)$. The overall time required to find the efficient frontier is then $O(L^2 n^4)$, which is polynomial, since solving the min-cost max-flow problem takes $O(n)$ time and since there are $O(L^2)$ classes.

Lemma 2.6 *Let j be any job. Between two successive events of type 2 job j gets fully compressed or decompressed at most n times.*

Proof. We look at the network corresponding to some value of Δ, where Δ is such that, either job j is compressed, or it will get compressed when Δ is increased by $\epsilon > 0$. Let $B(j)$ denote the set of jobs that are inspected for possible compression by our algorithm before job j. Consider the network at the moment that it is (j, t)'s turn to receive flow for the first time. Because of our assumption that job j is (about to be) compressed, we know that is no arc (i, t) with $i \in B(j)$ and $i < j$ that is not saturated.

Let e be the largest index with $e \leq j$ such that the arc $(s, e-1)$ is not saturated; if all arcs $(s, 1), \ldots, (s, j-1)$ are saturated, then $e \leftarrow 1$. For our next definition, we have to distinguish between two cases. If not all arcs $(s, j), \ldots, (s, n)$ are saturated, then let q denote the smallest index with $q \geq j$ such that the arcs $(s, q+1), \ldots, (s, n)$ are saturated; we put $q \leftarrow n$ if arc (s, n) is not saturated. If all arcs $(s, j), \ldots, (s, n)$ are saturated (which indicates that job j in fully decompressed), then we use the above definition of q on basis of the situation in the network where Δ is increased by ϵ; in that network, at least one of the arcs $(s, j), \ldots, (s, n)$ is not saturated, since job j would remain totally decompressed otherwise. Since the arc (s, q) is not saturated, we know that all arcs (i, t) with $i \leq q$ that contain flow must be saturated, and hence that the flow already floating through the arcs $(s, e), \ldots, (s, q)$ is equal to the sum U of the u_i values of the jobs $i \in B(j)$ with $e \leq i \leq q$; because of our choice of e and q, we know that this flow will remain the same if we increase Δ by ϵ. Arc (j, t) will serve the total remaining capacity of the arcs $(s,j),\ldots,(s,n)$, which is equal to the remaining capacity of the arcs $(s,e),\ldots,(s,n)$, and this spare capacity is equal to

$$b_q + v_q\Delta - (b_{e-1} - v_{e-1}\Delta) - U = b_q - b_{e-1} - U + \Delta(v_q - v_{e-1}).$$

This implies that, as long as e and q remain unchanged, x_j will be either constant, or will show monotonic behavior, where the type of behavior depends on the sign of $v_q - v_{e-1}$.

We now look at what is necessary to change either q or e. First, suppose that q changes; q can, either increase to r, or decrease to p. An increase of q to r signals that the arcs $(s,q+1),\ldots,(s,r)$ can no longer be saturated by the jobs in $B(j)$. Since the arc (s,q) is not saturated, the arcs $(s,q+1),\ldots,(s,r)$ are served by the jobs $i \in B(j)$ with $q+1 \le i \le r$, which used to be capable of saturating these arcs. Hence, the capacity of the arcs $(s,q+1),\ldots,(s,r)$, which is equal to $b_r - b_q + \Delta(v_r - v_q)$, must have gone up. Therefore, we must have that $v_r - v_q > 0$, that is $v_r > v_q$. A decrease of q to p on the other hand shows that now also the arcs $(s,p+1),\ldots,(s,q)$ can be saturated by the jobs in $B(j)$. This can only be due to a decrease of the capacity of the arcs $(s,p+1),\ldots,(s,q)$, which signals that $v_q - v_p < 0$, that is $v_p > v_q$. Hence, the index q can only be replaced by an index with larger v_j value.

Now supposes that either e increases to f, or decreases to d. In the first case, we know that the arc $(s,f-1)$, which was saturated before is not saturated anymore. This can only be the case if the capacity of the arcs $(s,e),\ldots,(s,f-1)$ has increased, which implies that $v_{f-1} > v_{e-1}$. Similarly, if e decreases to d, then the arcs $(s,d),\ldots,(s,e-1)$ get saturated, which implies a decrease in capacity of these arcs, from which we obtain that $v_{d-1} > v_{e-1}$. Hence, also in this case we see that the index $e-1$ can only be replaced by an index with larger v_j value. Since there are at most n possible changes, we see that x_j can change from increasing to decreasing or vice versa at most n times. ■

What is left to do is to bound the number of occurrences of events of type 2. Recall that the total amount of compression of the jobs $1,\ldots,j$ necessary to let job j meet its due date $\bar{d}_j - v_j\Delta$ is equal to

$$b_j(\Delta) \doteq \max\{0, \sum_{i=1}^{j} a_i - \bar{d}_j + v_j\Delta\};$$

in contrast to the description of our algorithm we have now added the argument Δ. In the description of our algorithm, we assumed that $b_1 \le b_2 \le \ldots \le b_n$, after which we could make the capacities of arc (s,j) equal to $b_j - b_{j-1}$, for $j = 1,\ldots,n$. Now we no longer assume that $b_1(\Delta) \le b_2(\Delta) \le \ldots \le b_n(\Delta)$. We incorporate this by putting the capacity c_{sj} of arc (s,j) equal to

$$c_{sj}(\Delta) \leftarrow \{0, b_j(\Delta) - \max_{1\le i\le j-1} b_i(\Delta)\} \qquad \forall j = 2,\ldots,n,$$

and $c_{s1}(\Delta) = b_1(\Delta)$. An event of type 2 occurs if c_{sj} either drops to zero, or emerges from zero. Define $a(j)$ $(j = 2,\ldots,n)$ as

$$a(j) \leftarrow \arg\max_{1\le i\le j-1} b_i(\Delta);$$

$a(j)$ can change, depending on Δ. But if $a(j)$ changes from, say index g to h, then this implies that the gradient v_g of the function $b_g(\Delta)$ must be smaller than the gradient v_h of the function $b_h(\Delta)$.

The capacity csj ($j = 2, \dots, n$) can drop to zero, only if $b_{a(j)}(\Delta)$ grows faster than $b_j(\Delta)$, that is, if $v_{a(j)} > v_j$, for $j = 2, \dots, n$. Moreover, a change of identity of $a(j)$ will not result in c_{sj} emerging from 0 again. It is possible, though, that c_{sj} becomes positive, but this can only be the case if has not been positive before. Hence, we have shown the following results.

Lemma 2.7 *The total number of events of type 2 is bounded by $O(n)$.* ■

Corollary 2.8 *The number of cornerpoints of the efficient frontier in the interval $(y_{k-1}, y_k]$ it at most equal to $O(n^3)$.* ■

Theorem 2.9 *Problem P4 for $1|contr|f_{\max}$ is solvable in $O(L^2n^4)$ time.* ■

3 Minimizing total weighted job completion time

In this section we discuss the single machine scheduling problem of minimizing total weighted job completion time with controllable job processing times. For every job J_j ($j = 1, \dots, n$), there is a corresponding positive integer weight w_j. All jobs are available for processing at time 0. If a job J_j completes at time C_j in some schedule, then it contributes w_jC_j to the objective function. This leads to the scheduling problem $1|contr|\sum w_jC_j$ with controllable processing times where

$$F_1(f, \pi, \vec{x}) \;=\; \sum_{j=1}^{n} w_{\pi(j)} \left(\sum_{i=1}^{j} a_{\pi(i)} - x_{\pi(i)} \right). \tag{4}$$

Observation 3.1 *There always exists an optimal solution to P1 for $1|contr|\sum w_jC_j$, in which every job is either totally uncompressed ($x_j = 0$) or totally compressed ($x_j = u_j$).*

Proof. Consider such an optimal solution, and assume that $0 < x_j < u_j$ holds for some job J_j. Let C_j denote the completion time of J_j in this schedule, and denote by W the total weight of all jobs that complete at time C_j or later. If we change x_j to $x_j + \delta$ for some δ with $-x_j \le \delta \le u_j - x_j$, then the objective value for P1 changes by $\delta(W - c_j)$. Now $\delta(W - c_j)$ is a linear function in δ, and hence it is minimized at one of the boundary points $\delta = -x_j$ or $\delta = u_j - x_j$. By repeating this argument for every single job, the claim follows. ■

Theorem 3.2 *Problem P1 for $1|contr|\sum w_jC_j$ is NP-hard.*

Proof. We reduce the NP-hard problem PARTITION to the scheduling problem. Let $e_1, \dots, e_n$ be a sequence of positive integers that sum up to $2E$. The question is to decide whether there exists an $I \subset \{1, \dots, n\}$ with $\sum_{i\in I} e_i = E$. For every e_j, we introduce a corresponding job J_j with parameters $a_j = u_j = e_j$ and $c_j = E + \frac{1}{2}e_j$, and with weight $w_j = e_j$. We claim that the constructed scheduling instance has a solution with value at most $\frac{3}{2}E^2 + \frac{1}{2}\sum_{i=1}^{n} e_i^2$ if and only if the PARTITION instance has a solution.

Suppose that the PARTITION instance has a solution I. Compress all jobs $i \in I$ to zero length and leave all remaining jobs uncompressed. The cost for compression then equals

$$\sum_{i\in I} c_i a_i \;=\; \sum_{i\in I} (E + \frac{1}{2}e_i)e_i \;=\; E^2 + \frac{1}{2}\sum_{i\in I} e_i^2.$$

The total weighted completion time of the uncompressed jobs equals

$$\frac{1}{2}(\sum_{i\notin I} e_i)^2 + \frac{1}{2}\sum_{i\notin I} e_i^2 \;=\; \frac{1}{2}E^2 + \frac{1}{2}\sum_{i\notin I} e_i^2.$$

Hence, the scheduling instance has a solution with value at most $\frac{3}{2}E^2 + \frac{1}{2}\sum_{i=1}^n e_i^2$.

Next, suppose that the scheduling instance has a solution with value at most $\frac{3}{2}E^2 + \frac{1}{2}\sum_{i=1}^n e_i^2$. By Observation 3.1, we may assume that every job is either totally compressed or totally uncompressed. Define I to be the set of totally compressed jobs. Define $\beta = \sum_{i\in I} e_i$. Then the objective value of P1 equals

$$\sum_{i\in I} c_i a_i + \frac{1}{2}(\sum_{i\notin I} a_i)^2 + \frac{1}{2}\sum_{i\notin I} a_i^2 \;=\; E\beta + \frac{1}{2}(2E-\beta^2) + \frac{1}{2}\sum_{i=1}^n e_i^2.$$

By our assumption, this expression is at most $\frac{3}{2}E^2 + \frac{1}{2}\sum_{i=1}^n e_i^2$. This is equivalent to $\frac{1}{2}(E-\beta)^2 \le 0$. Hence, $E=\beta$ must hold and the PARTITION instance indeed has a solution. ■

References

[1] B. ALIDAEE AND A. AHMADIAN [1993]. Two parallel machine sequencing problems involving controllable job processing times. *European Journal of Operational Research 70*, 335–341.

[2] T.C.E. CHENG, Z.-L. CHEN, AND C.-L. LI [1996]. Single-machine scheduling with trade-off between number of tardy jobs and resource allocation. *Operation Research Letters 19*, 237–242.

[3] M.R. GAREY AND D.S. JOHNSON [1979]. *Computers and Intractability: A Guide to the Theory of NP–Completeness.* Freeman, San Francisco.

[4] R.L. GRAHAM, E.L. LAWLER, J.K. LENSTRA, AND A.H.G. RINNOOY KAN [1979]. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics 5*, 287–326.

[5] A. JANIAK [1986]. Time-optimal control in a single machine problem with resource constraints. *Automatica 22*, 745–747.

[6] A. JANIAK AND M.Y. KOVALYOV [1996]. Single machine scheduling subject to deadlines and resource dependent processing times. *European Journal of Operational Research 94*, 284–291.

[7] E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN, AND D.B. SHMOYS [1993]. Sequencing and scheduling: Algorithms and complexity. In: S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin (eds.) *Logistics of Production and Inventory*, Handbooks in Operations Research and Management Science 4, North-Holland, Amsterdam, 445–522.

[8] E. NOWICKI AND S. ZDRZAŁKA [1990]. A survey of results for sequencing problems with controllable processing times. *Discrete Applied Mathematics 26*, 271–287.

[9] E. NOWICKI AND S. ZDRZAŁKA [1995]. A bicriterion approach to preemptive scheduling of parallel machines with controllable job processing times. *Discrete Applied Mathematics 63*, 237–256.

[10] F.M. RUIZ DIAZ AND S. FRENCH [1984]. A note on SPT scheduling of a single machine with controllable processing times. Note 154, Department of Decision Theory, University of Manchester.

[11] A.V. TUZIKOV84 [1984]. A two-criterion scheduling problem allowing for variation in job execution. *Zhurnal Vychistlitel'noj Matematiki i Matematicheskoj Fiziki 24*, 1585–1590.

[12] R.G. VICKSON [1980]. Two single machine sequencing problems involving controllable job processing times. *AIIE Transactions 12*, 258–262.

[13] R.G. VICKSON [1980]. Choosing the job sequence and processing times to minimize total processing plus flow cost on a single machine. *Operations Research 28*, 1155–1167.

[14] L.N. VAN WASSENHOVE AND K.R. BAKER [1982]. A bicriterion approach to time/cost tradeoffs in sequencing. *European Journal of Operational Research 11*, 48–54.