Approximation results for a bicriteria job scheduling problem on a single machine without preemption

Eric Angel, Evripidis Bampis, Laurent Gourvès

Approximation Results for a Bicriteria Job Scheduling Problem on a Single Machine without Preemption*

Eric Angel Evripidis Bampis Laurent Gourvès

LaMI, Université d'Évry, Tour Évry 2, 523 Place des terrasses de l'agora, 91000 Évry Cedex
{*angel,bampis,lgourves*}*@lami.univ-evry.fr*

Abstract

We study the problem of minimizing the average weighted completion time on a single machine under the additional constraint that the sum of completion times does not exceed a given bound B ($1|\sum C_j \leq B|\sum w_j C_j$) for which we propose a $(2,1)$-approximation algorithm. We also address the problem $1|\sum c_j C_j \leq B|\sum w_j C_j$ for which we present a $(2,2)$-approximation algorithm. After showing that the problem of minimizing two different sums of weighted completion times is intractable, we present an algorithm which computes a $(2(1+\varepsilon),1)$ (resp. $(2(1+\varepsilon),2)$)-approximate Pareto curve for the problem $1||(\sum C_j, \sum w_j C_j)$ (resp. $1||(\sum c_j C_j, \sum w_j C_j)$).

Keywords: Bicriteria optimization; Single machine scheduling; Approximate Pareto curve

1 Introduction

Because of their usefulness in industrial applications, scheduling problems are extensively studied since the middle of the 20^{th} century. More and more complex models are examined by researchers to fit to practical situations but solutions are often evaluated with respect to only one objective (or criterion). Nevertheless, the multi-objective nature of many of these problems created a new interest in this topic [6, 13].

In this paper, we consider the problem of scheduling n jobs on a single machine without *preemption* (i.e. the processing of a job cannot be interrupted once it has started). Each job $j \in \{1,\ldots,n\}$ has a processing time p_j, a weight w_j and a cost c_j. We assume that p_j, w_j and c_j are positive integers. Along the paper, we will consider that each job may start at time $t = 0$. Moreover, no precedence constraints are taken into account. For problems without release dates and non decreasing objective functions, it can be shown that only schedules without idle times have to be considered. Therefore it is sufficient to determine an optimal permutation of the jobs and, thus, the problem becomes a sequencing one. For ease of presentation, π will denote a solution (a permutation of the jobs). For any permutation π of n jobs, $\pi(i)$ denotes the i^{th} job of π (where $1 \leq i \leq n$) while $\pi^{-1}(j)$ denotes the position of job j in π (where $1 \leq j \leq n$). For a permutation π, the completion time of a job $j = \pi(i)$ is denoted by $C_j^\pi = \sum_{1 \leq k \leq i} p_{\pi(k)}$. When the context is clear, the superscript π will be dropped, i.e., C_j will be used instead of C_j^π. The addressed problem is bi-objective: along the article, the first (resp. second) objective will be called the *total cost* (resp. the *total weight*) of the schedule. We assume that $c(\pi)$ and $w(\pi)$ denote respectively the total cost and the total weight of a permutation π:

$$c(\pi) = \sum_{1 \leq j \leq n} c_j C_j^\pi,\ w(\pi) = \sum_{1 \leq j \leq n} w_j C_j^\pi.$$

*Research partially supported by the thematic network APPOL II (IST 2001-32007) of the European Union and the France-Berkeley Fund project MULT-APPROX

Dealing with different (often conflicting) objectives leads to the following statement: in general, there is not a unique optimal solution. The set of efficient solutions (called *Pareto curve*) constitutes a *trade-off* between the different objectives. Because of the *NP*-hardness and/or the *intractability* of most of the considered problems, designing an algorithm which computes in polynomial time a Pareto optimal solution or the whole *Pareto curve* is in general unlikely. Among the different classical approaches for multi-objective problems, one can distinguish the following two: The first one called the *Pareto approach* consists in computing a set of solutions which approximates the entire Pareto curve. The second one, called *budget approach*, consists in minimizing one criterion while the other does not exceed a given budget.

We consider these two approaches for our scheduling problem. Using the classical three fields notation, the budget version of our problem is denoted by $1|\sum c_jC_j \leq B|\sum w_jC_j$ (where B is the given budget). On the other hand, $1||(\sum c_jC_j, \sum w_jC_j)$ denotes the problem of computing a set of solutions which approximates the whole Pareto curve.

Previous work: With a reduction from PARTITION, Hoogeveen [6] shows that $1|\sum c_jC_j \leq B|\sum w_jC_j$ is *NP*-hard even if the cost of each job is fixed to one. Since the constrained problem is *NP*-hard, computing the whole exact Pareto curve is also *NP*-hard. However, it is possible to get in polynomial time a subset of the Pareto curve for $1||(\sum c_jC_j, \sum w_jC_j)$. Bagchi [3] proposes a combinatorial algorithm which outputs the set of all supported solutions[1] of the Pareto curve for a bicriteria scheduling problem on a single machine (the two objectives are the sum of completion times and the total absolute differences in completion times). As mentioned by Hoogeveen [6], this algorithm can be adapted to our problem. However, Bagchi [3] states that *"an algorithm for optimizing one criterion subject to a constraint on the other criterion should be quite valuable"*. This was the main motivation for our work.

More recently, Angel et al. [1] considered the *simultaneous approximation approach*[2] for the bicriteria problem of minimizing simultaneously the sum of completion times and the sum of weighted completion times. They proposed a $(1+\frac{1}{\gamma}, 1+\gamma)$-approximation algorithm for any $\gamma > 0$. They have also proved that for $0 < \gamma \leq 1$, there is an instance such that no (x,y)-schedule (x is the sum of completion times and y the sum of weighted completion times of the schedule) with $x < 1+\gamma$ and $y < 1+\frac{1-\gamma}{2\gamma+1}$ exists .

Our contribution: We study the problem $1|\sum c_jC_j \leq B|\sum w_jC_j$, and we provide two approximation algorithms.

Furthermore, we use these results in order to obtain an *approximate* Pareto curve for the problem $1||(\sum c_jC_j, \sum w_jC_j)$. We also show that the Pareto curve of the problem can contain an exponential number of solutions. We use techniques introduced in [3, 5, 9, 11].

The paper is organized as follows: We first present a combinatorial $(2,1)$-approximation algorithm for the problem $1|\sum C_j \leq B|\sum w_jC_j$. The next section addresses the problem $1|\sum c_jC_j \leq B|\sum w_jC_j$ for which we present a $(2,2)$-approximation algorithm based on linear programming. In Section 4.2, we show the intractability of problems $1||(\sum C_j, \sum w_jC_j)$ and $1||(\sum c_jC_j, \sum w_jC_j)$. Moreover, we present an algorithm to compute a $(2(1+\varepsilon),1)$-approximate (resp. $(2(1+\varepsilon),2)$-approximate) Pareto curve for the problem $1||(\sum C_j, \sum w_jC_j)$ (resp. $1||(\sum c_jC_j, \sum w_jC_j)$).

2 A constant approximation algorithm for $1|\sum C_j \leq B|\sum w_jC_j$

This section presents how the approach used in [11] can be adapted to the problem $1|\sum C_j \leq B|\sum w_jC_j$. The starting point is a formulation of the problem as an optimization program but the derived algorithm is

[1] Solutions of a Pareto curve are partitioned into two distinct sets: supported and non-supported Pareto optimal solutions. A solution of a bicriteria problem is supported iff it is optimal for a linear combination of the two criteria. Non-supported solutions are often much harder to compute.

[2] One tries to compute a single solution that is good, to an approximation with performance guarantee point of view, on each criterion. For example, this approach is followed by Aslam et al. [2] for a bicriteria scheduling problem.

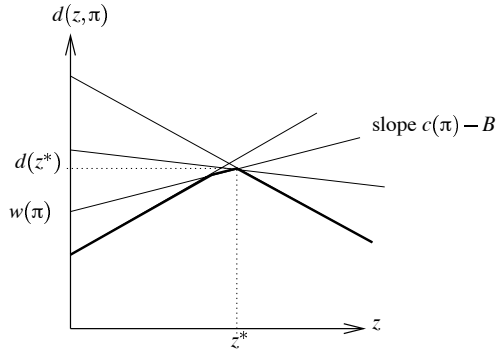

Figure 1: The plot of $d(z,\pi)$ as z varies while $d(z)$ is drawn in bold.

combinatorial. We first recall the definition of an (α,β)-approximate solution in our specific context before presenting the result and the method to get it.

Definition 1 *Given B, an (α,β)-approximate solution is a permutation with total cost at most αB and of total weight at most βW, where W is the minimum total weight of any permutation of total cost at most B.*

The result of this section is given in the following theorem.

Theorem 1 *There is a polynomial time algorithm for $1|\sum C_j \leq B|\sum w_j C_j$ which outputs a $(2,1)$-approximate solution.*

2.1 Lagrangean relaxation

Given a feasible cost B, the problem can be formulated as the following optimization program (c_j appears but in this section it is considered to be equal to one for each job j):

$$\begin{array}{rll} \text{Min}_\pi & \sum_{j=1}^n & w_j C_j^\pi \\ \text{subject to:} & & \\ & \sum_{j=1}^n & c_j C_j^\pi \leq B \\ & \pi & \text{is feasible.} \end{array}$$

Using Lagrangean relaxation consists in inserting the constraint $\sum_{j=1}^n c_j C_j^\pi \leq B$ into the cost function with a real positive multiplying factor z:

$$\begin{aligned} d(z,\pi) &= \sum_{j=1}^n (w_j + z c_j) C_j^\pi - zB, \\ d(z) &= \text{Min}_\pi\, d(z,\pi). \end{aligned}$$

One can see that $d(z)$ is a lower bound on W (the weight of an optimal solution for the optimization program). To provide the best lower bound on W, we try to find z^* such that $d(z^*) = \text{Max}_{z\geq 0}\, d(z)$. The function $d(z)$ is concave and piecewise linear (see Figure 1). Any permutation π corresponds to a line $d(z,\pi)$ with slope $c(\pi)-B$ and intercept $w(\pi)$. If we consider the monocriterion minimization problem $1||\sum(w_j + z^* c_j)C_j$, finding the best permutation of jobs can easily be done in polynomial time with Smith's rule [12]. Indeed, jobs must be scheduled by increasing $p_j/(w_j + z^* c_j)$ ratio. Moreover, several permutations can

Step 1: Compute $\pi_{\leq}$ and $\pi_{\geq}$
Step 2: Compute $PATH$ between $\pi_{\leq}$ and $\pi_{\geq}$
Step 3: Return the first permutation π_{out} in $PATH$ s.t. $c(\pi_{out}) \geq B$

Table 1: Sketch of the combinatorial algorithm.

HIERARCHICAL ORDER 1:
if $p_u/(w_u + z^{\#} c_u) < p_v/(w_v + z^{\#} c_v)$
then u is scheduled before v
else if $p_u/(w_u + z^{\#} c_u) > p_v/(w_v + z^{\#} c_v)$
then v is scheduled before u
else if $p_u/c_u \leq p_v/c_v$
then u is scheduled before v
else v is scheduled before u

HIERARCHICAL ORDER 2:
if $p_u/(w_u + z^{\#} c_u) < p_v/(w_v + z^{\#} c_v)$
then u is scheduled before v
else if $p_u/(w_u + z^{\#} c_u) > p_v/(w_v + z^{\#} c_v)$
then v is scheduled before u
else if $p_u/c_u \leq p_v/c_v$
then v is scheduled before u
else u is scheduled before v

Table 2: Hierarchical orders 1 and 2.

minimize $\sum(w_j + z^* c_j)C_j$ if there are jobs with equal $p_j/(w_j + z^* c_j)$ ratio. In this case, there must be at least one permutation $\pi_{\leq}$ such that $c(\pi_{\leq}) \leq B$ and at least another one $\pi_{\geq}$ such that $c(\pi_{\geq}) \geq B$. Just look at Figure 1, since z^* is optimal, at least two lines go through $d(z^*)$: one with a negative slope and another one with a positive slope. A line represents a permutation π whose slope is $c(\pi) - B$. If these two solutions do not exist, we have a contradiction to the fact that z^* is optimal. Since these two permutations both minimize $\sum(w_j + z^* c_j)C_j$, it is possible to swap specific adjacent jobs (namely, jobs with equal $p_j/(w_j + z^* c_j)$ ratio) without any increase in the cost.

2.2 The algorithm

Any permutation π_b can be reached from another one π_a by a series of swaps involving adjacent jobs[3]. Moreover, the number of intermediate permutations is polynomially bounded (at most $(n(n+1))/2$). Such a procedure can be employed to reach $\pi_{\geq}$ from $\pi_{\leq}$, moreover, swaps can be restricted to jobs with equal $p_j/(w_j + z^* c_j)$ ratio. As a consequence, each intermediate permutation has the same cost as $\pi_{\leq}$ and $\pi_{\geq}$. In the sequel, $PATH$ denotes such an ordered list of intermediate permutations between $\pi_{\leq}$ and $\pi_{\geq}$. A sketch of the algorithm is given in Table 1 while the next subsection provides a combinatorial procedure to compute $\pi_{\leq}$ and $\pi_{\geq}$.

2.3 How to get $\pi_{\leq}$ and $\pi_{\geq}$

For any fixed $z^{\#}$, it is possible to say whether $z^{\#} < z^*$, $z^{\#} = z^*$ or $z^{\#} > z^*$. Just consider the hierarchical order 1 defined in Table 2. With it, one can produce a permutation $\pi^{\star}$; if $c(\pi^{\star}) \geq B$ then $z^{\#} \leq z^*$. Indeed, among all optimal schedules for $w_j + z^{\#} c_j$, there is no permutation with total cost strictly lower than $c(\pi^{\star})$. Now consider the hierarchical order 2 defined in Table 2. With it, one can produce a permutation $\pi^{\star\star}$; if $c(\pi^{\star\star}) \leq B$ then $z^{\#} \geq z^*$. Symmetrically, there is no permutation with total cost strictly greater than $c(\pi^{\star\star})$. Therefore, if $c(\pi^{\star\star}) \leq B$ and $c(\pi^{\star}) \geq B$ then $z^{\#} = z^*$. In fact, determining whether $z^{\#}$ is smaller, larger or equal to z^* requires the computation of two permutations which can be done in $O(n \log n)$ time. In the sequel, we directly compute $\pi_{\leq}$ and $\pi_{\geq}$ without previously determining z^*. To do that, we exploit the fact that it is

[3] W.l.o.g, suppose that $\pi_b(i) = i$ for all i and apply a *Bubble sort* algorithm [7] on π_a.

Case $z_{uv} < z^*$**:** schedule u before v if $g(z_{uv}+\varepsilon) < h(z_{uv}+\varepsilon)$, otherwise schedule v before u.
Case $z_{uv} = z^*$**:** schedule u before v if $p_u/c_u \leq p_v/c_v$, otherwise schedule v before u.
Case $z_{uv} > z^*$**:** schedule u before v if $g(z_{uv}-\varepsilon) < h(z_{uv}-\varepsilon)$, otherwise schedule v before u.

Table 3: Three cases.

possible to know the ordering between any pair of jobs (u,v) without knowing z^* [11]. Just consider the two following functions of z:

$$g(z) = p_u/(w_u + z c_u),\ h(z) = p_v/(w_v + z c_v).$$

We distinguish two cases, when g and h have no breakpoint and when g and h have a breakpoint z_{uv}. For the first case, the ordering between u and v does not depend on z. Therefore, by the rule of Smith, u must be scheduled before v if $g(z) \leq h(z)$, otherwise schedule v before u. For the second case, the ordering between u and v is established by z^*; namely, it changes depending if the value of z^* is smaller or larger than z_{uv}. Determining whether z_{uv} is smaller or larger than z^* requires the computation of two permutations ($O(n \log n)$ with Smith's rule). Now we consider three cases (see Table 3) for any arbitrary small value ε. A procedure can be derived to output $\pi_{\leq}$ (to get $\pi_{\geq}$, in Table 3, replace the second case by **Case** $z_{uv} = z^*$**:** *schedule u before v if* $p_u/c_u \geq p_v/c_v$*, otherwise schedule v before u*). It needs the computation of z_{uv} for each pair of distinct jobs and for each positive z_{uv}, the comparison to z^* requires the computation of two optimal permutations. Thus, the time required to compute $\pi_{\leq}$ or $\pi_{\geq}$ is $O(n^3 \log n)$.

2.4 Proof of Theorem 1

Swapping any pair of adjacent jobs of a permutation π can induce a variation of its total cost denoted by δl. We assume that π contains two jobs u and v such that u is placed just before v. We denote by p_u, C_u and c_u (resp. p_v, C_v and c_v) the processing time, completion time and cost of u (resp. v). The total cost of π before and after the swap are respectively denoted by $c(\pi)$ and $c'(\pi)$. One can see that $\delta l = c'(\pi) - c(\pi) = c_u(C'_u - C_u) + c_v(C'_v - C_v) = c_u p_v - c_v p_u$ since $C'_u = C_u + p_v$ and $C'_v = C_v - p_u$. Therefore, δl can be upper bounded as follows:

$$\delta l \leq \mathrm{Max}_{u,v}\{c_u p_v - c_v p_u\} \leq c_{max} C_{max},$$

where $c_{max} = \mathrm{Max}_j\{c_j\}$ and $C_{max} = \mathrm{Max}_j\{p_j\}$.

Now, consider the algorithm whose sketch is given in Table 1. It outputs a permutation (say π_{out}) which is the first in $PATH$ having a total cost larger than B. Therefore, one can bound $c(\pi_{out})$ by $B + \delta l$. We get:

$$c(\pi_{out}) \leq B + \delta l \leq B + c_{max} C_{max} \leq (1 + c_{max})B.$$

We can now argue that C_{max} must be at most B since otherwise the instance will be infeasible. Since $w(\pi_{out}) \leq d(z^*) \leq W$, π_{out} approximates (B,W) within a factor $(1 + c_{max}, 1)$. Moreover, since for all j, $c_j = 1$, we get a $(2,1)$-approximation. □

With this former algorithm, the approximation is not constant when, for each job, c_j is not fixed to one. However, the next section presents a constant approximation algorithm based on linear programming.

3 A constant approximation algorithm for $1|\sum c_j C_j \leq B|\sum w_j C_j$

We are going to define a linear program whose optimal solution will allow us to build a $(2,2)$-approximate schedule by a simple heuristic procedure. The same technique was used by Hall et al. [5] to obtain a 2-approximation algorithm for the problem denoted by $1|prec|\sum_j w_j C_j$ which consists in scheduling jobs on

a single machine to minimize the sum of weighted completion times while jobs are subject to precedence constraints.

Theorem 2 *There is a (2,2)-approximation algorithm for* $1|\sum c_j C_j \leq B|\sum w_j C_j$.

Proof. It follows from Smith's rule that if we set $w_j = p_j$ for all jobs j, then the sum $\sum_{j=1}^{n} w_j C_j^{\pi}$ is invariant for any ordering π of the jobs. Therefore we have $\sum_{j=1}^{n} p_j C_j^{\pi} = \sum_{j=1}^{n} p_j(\sum_{i=1}^{j} p_i) = \sum_{j=1}^{n}\sum_{i=1}^{j} p_j p_i$. Let y_j be a decision variable which represents the completion time of job j for $j = 1,\ldots,n$. We have therefore for any feasible schedule, $\sum_{j=1}^{n} p_j y_j \geq \sum_{j=1}^{n}\sum_{i=1}^{j} p_j p_i$ (the inequality comes from the possibility of idle time in the schedule). It can be easily seen that this inequality remains valid for any subset $A \subseteq \{1,\ldots,n\}$ of jobs. We obtain that $\sum_{j\in A} p_j y_j \geq \frac{1}{2}\left[\sum_{j\in A} p_j^2 + (\sum_{j\in A} p_j)^2\right]$ is a valid inequality for the completion times of jobs in any feasible schedule. The linear program we consider is the following one:

$$\begin{aligned}
\text{Min} \quad & \sum_{j=1}^{n} w_j y_j && (1)\\
\text{subject to:} \quad & \sum_{j\in A} p_j y_j \geq \frac{1}{2}\left[\sum_{j\in A} p_j^2 + (\sum_{j\in A} p_j)^2\right], \quad \forall A \subseteq \{1,\ldots,n\} && (2)\\
& \sum_{j=1}^{n} c_j y_j \leq B && (3)\\
& y_j \geq 0, \qquad 1 \leq j \leq n && (4)
\end{aligned}$$

Queyranne [10] has shown that the linear program given by (1), (2) and (4) is solvable in polynomial time. There exists a polynomial-time separation algorithm for the exponential number of constraints in the set (2), and by the equivalence between optimization and separation [4] the result follows. Adding the single constraint (3) to the set (2) does not prevent the separation algorithm to work, and therefore the linear program given by (1), (2), (3) and (4) remains polynomially solvable.

We solve this linear program to obtain an optimal solution $\tilde{y_j}$ for all j. These values do not generally represent a feasible schedule but are lower bounds for the completion time of jobs in an optimal solution. To obtain a feasible solution we schedule the jobs according to these values.

Let us assume that jobs have been renumbered so that $\tilde{y_1} \leq \ldots \leq \tilde{y_n}$. Let $\overline{C_j} = \sum_{k=1}^{j} p_k$ be the completion time of job j in the schedule we obtain, and let C_j^* be the completion time of job j in an optimal schedule.

Then $\tilde{y_j}\sum_{k=1}^{j} p_k \geq \sum_{k=1}^{j} p_k \tilde{y_k} \geq \frac{1}{2}\sum_{k=1}^{j} p_k^2 + \frac{1}{2}(\sum_{k=1}^{j} p_k)^2$ because of constraints (2). Therefore $\tilde{y_j}\sum_{k=1}^{j} p_k \geq \frac{1}{2}(\sum_{k=1}^{j} p_k)^2$, and $\tilde{y_j} \geq \frac{1}{2}\sum_{k=1}^{j} p_k = \frac{1}{2}\overline{C_j}$. We get that $\overline{C_j} \leq 2\tilde{y_j} \leq 2C_j^*$ and therefore $\sum_j w_j\overline{C_j} \leq \sum_j 2w_j C_j^*$. Moreover $\sum_j c_j\overline{C_j} \leq 2\sum_j c_j\tilde{y_j} \leq 2B$ because of constraints (3). □

The same idea for the bound of twice the LP value has been used earlier in [8].

Next section is devoted to the approximation of the Pareto curve with performance guarantee on the two criteria.

4 Approximating the whole Pareto curve

As stated by Hoogeveen [6], the whole set of supported solutions of the Pareto curve for our problem can be computed in polynomial time. At the same time, the problem is *NP*-hard, meaning that the hardness comes from the computation of the non-supported Pareto optimal solutions. Moreover, as we will see in Subsection 4.1, the whole Pareto curve can contain an exponential number of solutions. The set of supported Pareto optimal solutions constitutes an approximation of the whole Pareto curve. Nevertheless, we are interested in obtaining a worst case performance guarantee on both objectives. We are using for that the notion of $(\alpha,\beta)-$*approximate Pareto curve* (a general definition of an approximate Pareto curve with performance guarantee can be found in [9]).

Definition 2 *An (α,β)-approximate Pareto curve for our problem is a set P of permutations having the following property: For any permutation π^* in the exact Pareto curve P^*, there exists at least one permutation $\pi \in P$ such that $c(\pi) \leq \alpha\, c(\pi^*)$ and $w(\pi) \leq \beta\, w(\pi^*)$.*

4.1 Intractability

Suppose $1|\sum c_jC_j \leq B|\sum w_jC_j$ is solvable in polynomial time. Therefore, one can compute the whole Pareto curve by solving $1|\sum c_jC_j \leq B|\sum w_jC_j$ for each possible value of B. However, in the case where the Pareto curve contains an exponential number of solutions, this procedure has a complexity in time which is not polynomially bounded. We prove that, even if c_j is fixed to 1 for all jobs, the problem $1||(\sum c_jC_j, \sum w_jC_j)$ remains intractable[4].

We consider the following instance I_n with n jobs: each job j has a processing time n^{j-1}, a weight n^j-1 and a cost equal to 1.

Lemma 1 *For the instance I_n, $\sum_{1\leq j\leq n} c_jC_j + \sum_{1\leq j\leq n} w_jC_j$ is invariant for any ordering π of the jobs.*

Proof. Consider optimizing the expression: $\sum(c_jC_j + w_jC_j) = \sum((c_j+w_j)C_j)$ for the specific instance I_n. Each job has $p_j = n^{j-1}$ and $c_j + w_j = n^j$. Therefore $p_j/(c_j+w_j) = 1/n$ for all jobs. Thus by Smith's rule, any ordering π of the jobs is optimal. □

Lemma 2 *Let π and π' be two permutations of I_n. One has*

$$\sum_{i=1}^{n} C^{\pi}_{\pi(i)} = \sum_{i=1}^{n} C^{\pi'}_{\pi'(i)} \Leftrightarrow \forall i \in \{1,\ldots,n\}, \quad \pi(i) = \pi'(i).$$

Proof. ($\Leftarrow$) is obvious. To prove ($\Rightarrow$), we first state that the job n with processing time n^{n-1} (the largest in I_n) is at the same position in π and in π'. Let $i^\flat$ and $i^\#$ be positions such that $\pi(i^\flat) = \pi'(i^\#) = n$. Suppose that $i^\flat < i^\#$ (the case $i^\flat > i^\#$ is symmetric and can be treated by similar arguments) and $\sum_{i=1}^{n} C^{\pi}_{\pi(i)} = \sum_{i=1}^{n} C^{\pi'}_{\pi'(i)}$. The cost of any permutation π in I_n can be expressed as follows:

$$\begin{aligned}
\sum_{i=1}^{n} C^{\pi}_{\pi(i)} &= \sum_{i=1}^{n} \Big[\sum_{k=1}^{i} p_{\pi(k)}\Big] \\
&= n\,p_{\pi(1)} + (n-1)\,p_{\pi(2)} + \cdots + 2p_{\pi(n-1)} + p_{\pi(n)} \\
&= \sum_{i=1}^{n} (n-i+1) n^{\pi(i)-1}.
\end{aligned}$$

Let $\alpha_i = n-i+1$. Thus, $\alpha_{i^\flat}$ and $\alpha_{i^\#}$ are such that:

$$\sum_{i=1}^{n} C^{\pi}_{\pi(i)} = \alpha_{i^\flat} n^{n-1} + \sum_{\substack{1\leq i\leq n \\ i\neq i^\flat}} \alpha_i n^{\pi(i)-1} \text{ and } \sum_{i=1}^{n} C^{\pi'}_{\pi'(i)} = \alpha_{i^\#} n^{n-1} + \sum_{\substack{1\leq i\leq n \\ i\neq i^\#}} \alpha_i n^{\pi'(i)-1}.$$

Since $i^\flat < i^\#$, we have $\alpha_{i^\flat} > \alpha_{i^\#}$. One can show the following inequality:

$$\sum_{1\leq i\leq n} \alpha_i n^{\pi(i)-1} > \sum_{1\leq i\leq n} \alpha_i n^{\pi'(i)-1},$$

contradicting the fact that the total cost of π is equal to the cost of π'. Just consider the following inequalities which are true since $\forall i \in \{1,\ldots,n\}$, $1 \leq \alpha_i \leq n$:

[4] A problem is intractable if there exists at least one polynomially sized instance for which, no polynomial time algorithm can output its Pareto set.

HIERARCHICAL ORDER 3:
if $p_u/c_u < p_v/c_v$
then schedule u before v
else if $p_v/c_v < p_u/c_u$
then schedule v before u
else if $p_u/w_u \leq p_v/w_v$
then schedule u before v
else schedule v before u

HIERARCHICAL ORDER 4:
if $p_u/w_u < p_v/w_v$
then schedule u before v
else if $p_v/w_v < p_u/w_u$
then schedule v before u
else if $p_u/c_u \leq p_v/c_v$
then schedule u before v
else schedule v before u

Table 4: Hierarchical orders 3 and 4.

$$\begin{array}{rcl}
(\alpha_{i^\flat} - \alpha_{i^\#})n^{n-1} & \geq & \alpha_{\pi'^{-1}(n-1)}n^{n-2} \\
\alpha_{\pi^{-1}(n-1)}n^{n-2} & \geq & \alpha_{\pi'^{-1}(n-2)}n^{n-3} \\
 & \vdots & \\
\alpha_{\pi^{-1}(2)}n^1 & \geq & \alpha_{\pi'^{-1}(1)}n^0 \\
\alpha_{\pi^{-1}(1)}n^0 & > & 0
\end{array}$$

$$\sum_{\substack{1\leq i\leq n \\ i\neq i^\flat}} \alpha_i n^{\pi(i)-1} + (\alpha_{i^\flat} - \alpha_{i^\#})n^{n-1} \quad > \quad \sum_{\substack{1\leq i\leq n \\ i\neq i^\#}} \alpha_i n^{\pi'(i)-1}$$

It leads to say that $i^\flat$ cannot be different from $i^\#$. Now, using iteratively this argument on jobs of smaller processing time, one can show that two permutations of equal cost in I_n must be identical. □

Theorem 3 *The problem* $1||(\sum C_j, \sum w_j C_j)$ *is intractable.*

Proof. With Lemma 1 and 2, one can deduce that the Pareto curve of I_n is composed of $n!$ different points and given that the size of I_n is $O(n^2\, log(n))$, the result follows. □

4.2 Approximate Pareto curves for $1||(\sum c_j C_j, \sum w_j C_j)$

A Pareto curve is a set of solutions in which we can distinguish particular ones: the *extreme* points. On Figure 2, the two extreme points of the Pareto curve are denoted by A and B. In concrete terms, A has the lowest total weight among all optimal schedules for the total cost. Symmetrically, B has the lowest total cost among all optimal schedules for the total weight. Necessarily, these extreme points are supported. Using the hierarchical orders given in Table 4, one can compute them in $O(n \log n)$ time.

Suppose the cost-space is divided into columns (see Figure 2) in such a way that, for any couple of permutations (π, π') which are in the same column, one has $c(\pi) \leq (1+\varepsilon)c(\pi')$ with $\varepsilon > 0$. So, using an (α, β)-approximation algorithm for the problem $1|\sum c_j C_j \leq (1+\varepsilon)^m|\sum w_j C_j$ produces a solution which constitutes an $(\alpha(1+\varepsilon), \beta)$-approximation of solutions whose costs are between $(1+\varepsilon)^{m-1}$ and $(1+\varepsilon)^m$ (and particularly Pareto optimal ones). By repeating this procedure for $m = 0, 1 \ldots, M$ where $(1+\varepsilon)^M$ is greater than the total cost of B, we are able to compute an $(\alpha(1+\varepsilon), \beta)$-approximate Pareto curve of the problem. Since we consider a polynomially sized amount of budgets, this approximate Pareto curve can be computed in polynomial time.

With results from Sections 2 and 3 we derive the two following propositions:

Proposition 1 *There is a polynomial time algorithm for* $1||(\sum C_j, \sum w_j C_j)$ *which outputs a* $(2(1+\varepsilon), 1)$*-approximate Pareto curve.*

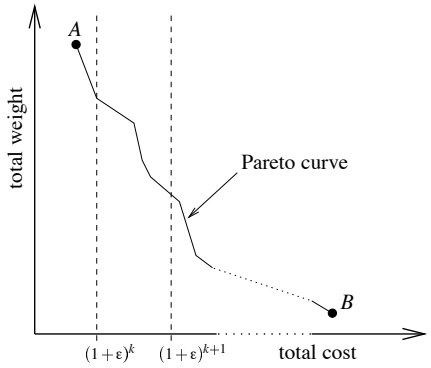

Figure 2: Dots A and B are the extreme points of the Pareto curve.

Proposition 2 *There is a polynomial time algorithm for* $1||(\sum c_jC_j, \sum w_jC_j)$ *which outputs a* $(2(1+\varepsilon),2)$*-approximate Pareto curve.*

5 Concluding remarks

In this paper we propose a $(2,1)$-approximation algorithm for the problem $1|\sum C_j \leq B|\sum w_jC_j$ and a $(2,2)$-approximation algorithm for the problem $1|\sum c_jC_j \leq B|\sum w_jC_j$. We also state intractability of problems $1||(\sum C_j, \sum w_jC_j)$ and $1||(\sum c_jC_j, \sum w_jC_j)$ for which we present a procedure to get approximate Pareto curves.

Note that it is also possible to get in polynomial time a $(2,1)$-approximate Pareto curve for $1||(\sum C_j, \sum w_jC_j)$ by taking advantage of Bagchi's approach [3]. Remark that, in Section 2, permutations denoted by $\pi_{\leq}$ and $\pi_{\geq}$ are always supported Pareto optimal solutions. Therefore, a $(2,1)$-approximate Pareto curve for the problem $1||(\sum C_j, \sum w_jC_j)$ can also be computed with the approximation algorithm of Section 2.

Remark that it is possible to get a $(2,2)$-approximation algorithm for the problem $1|\sum c_jC_j \leq B|\sum w_jC_j$ for which precedence constraints are added. Indeed, adding precedence constraints into the linear program of Section 3 does not prevent the heuristic to work.

Acknowledgment

We thank Aleksei Fishkin for pointing to us the reference [5].

References

[1] E. Angel, E. Bampis, and A.V. Fishkin. A note on scheduling to meet two min-sum objectives. In *Proceedings of the ninth international workshop on project management and scheduling (PMS'2004)*, pages 143–146, Nancy, France, 2004.

[2] J. Aslam, A. Rasala, C. Stein, and N. Young. Improved bicriteria existence theorems for scheduling. In *Symposium on Discrete Algorithms (SODA) Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 846–847, 1999.

[3] U. Bagchi. Simultaneous minimization of mean and variation of flow time and waiting time in single machine systems. *Operations Research*, 37:118–125, 1989.

[4] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequenses in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[5] L.A. Hall, A.S. Schulz, D.B. Shmoys, and J. Wein. Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.

[6] H. Hoogeveen. *Single-Machine Bicriteria Scheduling*. PhD thesis, Eindhoven University of Technology, 1992.

[7] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison Wesley, 1998.

[8] F. Margot, M. Queyranne, and Y. Wang. Decompositions, network flows, and a precedence constrained single-machine scheduling problem. *Operations Research*, 51(6):981–992, 2003.

[9] C.H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proc. of the 41th Annual IEEE symposium on Foundations of Computer Science*, pages 86–92, 2000.

[10] M. Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58:163–185, 1993.

[11] R. Ravi and M. Goemans. The constrained minimum spanning tree problem. In *Proc. of the Scandinavian Workshop on Algorithmic Theory (SWAT)*, pages 66–75, 1996.

[12] W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistic Quaterly*, 3:59–66, 1956.

[13] V. T'kindt and J-C. Billaut. *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer, 2002.