A note on scheduling to meet two min-sum objectives

Eric Angel, Evripidis Bampis, Aleksei V. Fishkin

A Note on Scheduling to Meet Two Min-Sum Objectives*

Eric Angel[1] Evripidis Bampis[1] Aleksei V. Fishkin[2]

1. LaMI, CNRS UMR 8042, Université d'Évry, France
{angel, bampis}@lami.univ-evry.fr
2. University of Kiel, Olshausenstr. 40, 24118 Keil, Germany
avf@informatik.uni-kiel.de

Abstract

We consider the problem of scheduling a set of independent jobs on a single machine where a solution is evaluated with respect to two min-sum objective functions: the sum of completion times and the sum of weighted completion times. In particular, we are interested in (α,β)-approximate schedules, i.e., schedules which are simultaneously at most α times from the optimum for the first objective, and β times from the optimum for the second objective. We propose a simple polynomial time $(1+\frac{1}{\gamma},1+\gamma)$-approximation algorithm which for any $\gamma>0$ always outputs $(1+\frac{1}{\gamma},1+\gamma)$-approximate schedules. In addition, we show that for $\gamma>1$, it is not possible to get an (x,y)-approximation algorithm with $1<x<1+\frac{1}{\gamma}$ and $1<y<1+\frac{\gamma-1}{2+\gamma}$.

1 Introduction

In recent years, a lot of attention has been devoted to solving multi-objective optimization problems, i.e., problems where the solutions are evaluated with respect to more than one objectives [2]. In particular, several results have been obtained for bi-objective scheduling problems which involve one min-max and one min-sum objective functions. Stein and Wein [5] considered two of the most popular ones: *the makespan* and *the sum of weighted completion times*. They proved that there exist schedules which are simultaneously at most two times from the optimum makespan and at most two times from the optimum sum of weighted completion times. The idea of their proof applies to a large class of scheduling problems, including scheduling of jobs on parallel and unrelated machines, preemptive scheduling, scheduling in the presence of precedence constraints, etc. Following this result, Aslam et al. [1] improved the bounds, and Rasala et al. [3] generalized it for all pairs of objectives in which the first one is among *maximum flow time*, *makespan*, or *maximum lateness*, and the second one is among *average flow time*, *sum of completion times*, or *number of on-time jobs*.

In this paper we consider the bi-objective problem of scheduling a set of independent jobs on a single machine to meet two min-sum objectives: the *sum of completion times* and the *sum of weighted completion times*. More formally, we are given a set of independent jobs $J=\{1,2,\ldots,n\}$. Each job j ($j=1,\ldots,n$) has a processing time p_j and a weight w_j. A schedule is obtained by sequencing the jobs on the machine in some order. There are two objectives: the sum of completion times $\sum_j C_j$ and the sum of weighted completion times $\sum_j w_jC_j$, where C_j denotes the completion time of job j. We are interested in finding (α,β)-approximate schedules, i.e., schedules which are at most α times from the optimum for $\sum_j C_j$ and at most β times from the optimum for $\sum_j w_jC_j$.

There are two well-known classical results in scheduling theory concerning our framework. The first one states that the optimum for the problem of minimizing the sum of completion times is obtained by sequencing the jobs in the shortest processing time (SPT) order, i.e., in non-decreasing order of p_j. The second one states that the optimum for the problem of minimizing the sum of weighted completion times is obtained by sequencing the jobs in Smith's order, i.e., in non-increasing order of w_j/p_j [4]. Indeed, an optimal schedule for each of the two objectives can be found in polynomial time. In general, we cannot expect a schedule to be simultaneously optimal for both objectives. Hence, we are interested in constructing an (α,β)-approximation algorithm, i.e., an algorithm that always produces (α,β)-approximate solutions in polynomial time.

Here we present a simple approximation algorithm, called the γ-Algorithm, which for any $\gamma>0$ outputs $(1+\frac{1}{\gamma},1+\gamma)$-approximate schedules in polynomial time. In addition, we give an example which demonstrates

*Supported by EU-project CRESCCO IST-2001-33135.

that for any $\gamma > 1$ there is an instance such that no (x,y)-schedule with $x < 1 + \frac{1}{\gamma}$ and $y < 1 + \frac{\gamma-1}{2+\gamma}$ exists. Interestingly, if we consider the case when $\gamma \to \infty$, the bounds tend to different values. Indeed, the example gives $(1,2)$ and the γ-Algorithm gives $(1,\infty)$. We conjecture that there exists no constant $\beta > 1$ such that $(1+\frac{1}{\gamma},\beta)$-approximate schedules can be found in polynomial time for any given $\gamma > 0$.

The rest of the paper is organized as follows. In the next section we introduce some notations and we point out some preliminary results. In Section 3, we informally sketch our approach, we give an outline of the γ-Algorithm and we analyze its performance. In Section 4 we give some non-existence bounds and in the last section we give some concluding remarks.

2 Preliminaries

Notations. To simplify the presentation, we assume that a schedule for the jobs in J is given as an arbitrary permutation $\sigma = (\sigma(1),\sigma(2),\ldots,\sigma(n))$, and use σ^{-1} to denote the inverse permutation of σ. Informally, this means that in schedule σ, each job $\sigma(j)$ ($j = 1,\ldots,n$) is on the j-th position and each job k ($k = 1,\ldots,n$) is on the $\sigma^{-1}(k)$-th position. In addition, we will use id to denote the identical permutation on set J, i.e., $id = (1,2,\ldots,n)$. Then, it holds that

$$\sigma \circ \sigma^{-1} = id \text{ and } \sigma^{-1} \circ \sigma = id.$$

Following the scheduling problem, we sequence the jobs of J on the machine in the order defined by σ. In this case, the completion time of each job j ($j = 1,\ldots,n$) in the schedule is defined as follows

$$C_j(\sigma) = \sum_{t=1}^{\sigma^{-1}(j)} p_{\sigma(t)} = \sum_{k:\sigma^{-1}(k)\le\sigma^{-1}(j)} p_k. \tag{1}$$

Then, the sum of job completion times in schedule σ is equal to

$$S(\sigma) = \sum_{j=1}^{n} C_j(\sigma) \tag{2}$$

and the sum of weighted job completion times in schedule σ is equal to

$$W(\sigma) = \sum_{j=1}^{n} w_j C_j(\sigma). \tag{3}$$

Throughout the paper we assume, w.l.o.g., that the jobs in J are numbered by $1,2,3,\ldots,n$ such that

$$p_1 \le p_2 \le p_3 \le \cdots \le p_n. \tag{4}$$

In addition, we define π to be a permutation on set J such that

$$\frac{w_{\pi(1)}}{p_{\pi(1)}} \ge \frac{w_{\pi(2)}}{p_{\pi(2)}} \ge \frac{w_{\pi(3)}}{p_{\pi(3)}} \ge \ldots \ge \frac{w_{\pi(n)}}{p_{\pi(n)}}. \tag{5}$$

Then, the optimum for the sum of completion times, denoted as $OPT1$, can be obtained by sequencing the jobs in the SPT order id:

$$OPT1 = \sum_{j=1}^{n} C_j(id) = \sum_{j=1}^{n}\sum_{z=1}^{j} p_z, \tag{6}$$

and the optimum for the weighted sum of completion times, denoted as $OPT2$, can be obtained by sequencing the jobs following Smith's order π:

$$OPT2 = \sum_{j=1}^{n} w_j C_j(\pi). \tag{7}$$

Given $\alpha,\beta > 0$, we look for a schedule σ such that

$$S(\sigma) \le \alpha \cdot OPT1 \text{ and } W(\sigma) \le \beta \cdot OPT2. \tag{8}$$

3 The γ-Algorithm

Our Approach to Approximation. Given jobs $1,2,\ldots,n$ and π, we can find $OPT1$ and $OPT2$ schedules, respectively. Our approach is to merge these two schedules into one, which is simultaneously near $OPT1$ and $OPT2$. We can informally sketch this as follows.

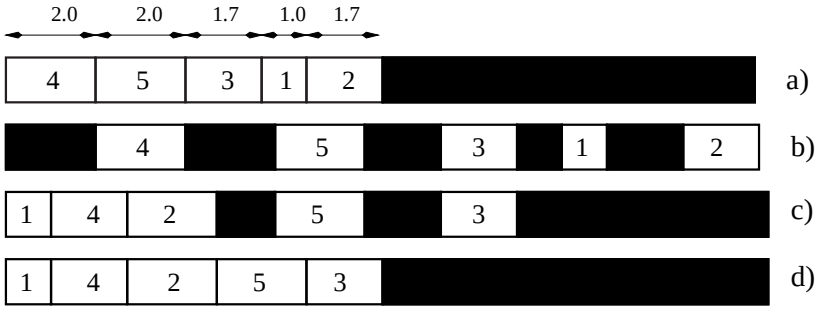

Figure 1: An illustration of the γ-Algorithm. The processing times of the jobs are $p_1 = 1$, $p_2 = p_3 = 1.7$ and $p_4 = p_5 = 2$. In the resulting schedule in d) jobs $4,5$ are large, jobs $1,2$ are small, and job 3 is medium.

First, we place all jobs on the machine as it is defined by π. For an illustration see Figure 1 a), where five jobs are placed by $\pi = (4,5,3,1,2)$. Next, we take $\gamma > 0$. We multiply all job completion times by a factor of $(1+\gamma)$ and move all start times to match without changing the job processing times. Then, each job j ($j = 1,\ldots,n$) increases its completion time by a factor of $(1+\gamma)$, and creates an idle "gap" (on its left) on the machine of size $\gamma \cdot p_j$. Job j is called the *owner* of the corresponding created gap on its left. For an illustration see Figure 1 b), where $\gamma = 1$.

Now we use these idle "gaps" to reschedule jobs. We select the job with the smallest *id* index and try to place it as early as possible, see job 1 in Figure 1 c). If there is enough idle time in the gap, then we "move" and "complete" this job. We will call such a "moved" job as *small*. Otherwise, we "shift" the job with the smallest π index to the left side "collecting" its idle time, for instance see job 4 in Figure 1 c). We will call such a "shifted" job as *large*. Next, we repeat this procedure among the "remaining" jobs. We always try to place the job with the smallest *id* index in the front of the job with the smallest π index, see jobs 2 and 5 in Figure 1 c), d). Indeed, it can happen that some jobs should stay at their positions: either there is not enough idle time on the machine or the job has the smallest index in both *id* and π, see job 3 in Figure 1 c), d). We will call such a "small-large" job as *medium*.

From one side, all jobs can only decrease their completion times. This means that the sum of weighted job completion times of is at most

$$(1 + \gamma) \cdot OPT2.$$

From another side, the "delay" of any job j in σ is formed by large jobs ℓ placed before j, and these large jobs cannot create enough time on the machine to "accommodate" jobs $1,2,\ldots,j-1,j$, i.e.

$$\gamma \cdot \sum p_\ell \;\leq\; \sum_{z \leq j} p_z \;=\; C_j(id).$$

This means that the sum of completion times is at most

$$\left(1 + \frac{1}{\gamma}\right) \cdot OPT1.$$

Therefore, we obtain a $(1+\frac{1}{\gamma}, 1+\gamma)$-schedule.

The γ-Algorithm. Now we can give an outline of our algorithm:

THE γ-ALGORITHM
Input: A parameter $\gamma > 0$, a job set J, schedules π and id.
Output: A schedule σ.

Step 1 Sequence the jobs of J on the machine following π order.
Step 2 Multiply all job completion times by a factor of $(1+\gamma)$ and move all jobs to the right to match their new completion times without changing their processing times.
Step 3 Fetch the jobs in id order and place them into the left as much as possible, in such a way that no job is delayed and as soon as a gap is declared too small it is closed by shifting the corresponding owner job to the left.

In that follows we prove the main result:

Theorem 1. *For any $\gamma > 0$, the γ-Algorithm is a $(1+\frac{1}{\gamma}, 1+\gamma)$-approximation algorithm.*

Relative Job Positions. By the γ-Algorithm, jobs preserve their relative positions:

Lemma 1. *In schedule σ, small and medium jobs follow id order whereas large and medium jobs follow π order.*

The Sum of Weighted Job Completion Times. Now we can prove the following result:

Lemma 2. *In schedule σ, the completion time of any job j ($j = 1,\ldots,n$) can be bounded as follows*

$$C_j(\sigma) \;\leq\; (1+\gamma)\cdot C_j(\pi). \tag{9}$$

Hence, the sum of the weighted job completion times is

$$W(\sigma) \;\leq\; (1+\gamma)\cdot OPT2. \tag{10}$$

Proof. For any job j ($j = 1,\ldots,n$) in σ, all jobs placed before j form two sets: J_j of large and medium jobs, and J'_j of small jobs. Then, the completion time of job j in σ is defined as follows

$$C_j(\sigma) \;=\; \left(p_j \;+\; \sum_{k\in J_j} p_k\right) \;+\; \sum_{s\in J'_j} p_s. \tag{11}$$

Since all medium and large jobs k in J_j follow π order, the completion time of job j in schedule π can be bounded as follows

$$C_j(\pi) \;\geq\; p_j \;+\; \sum_{k\in J_j} p_k. \tag{12}$$

Since each small job s in J'_j occurs only if there is enough idle time, the "delay" of job j in schedule σ can be bounded by

$$\sum_{s\in J'_j} p_s \;\leq\; \gamma\cdot\left(p_j \;+\; \sum_{k\in J_j} p_k\right). \tag{13}$$

Combining (11), (12) and (13) we prove (9). Thus, summing up over all jobs we prove (10). □

The Sum of Job Completion Times. Similarly, we can prove the following result:

Lemma 3. *In schedule σ, the completion time of any job j ($j = 1,\ldots,n$) can be bounded as follows*

$$C_j(\sigma) \;\leq\; \left(1+\frac{1}{\gamma}\right)\cdot C_j(id). \tag{14}$$

Hence, the sum of job completion times

$$S(\sigma) \;\leq\; \left(1 \;+\; \frac{1}{\gamma}\right)\cdot OPT1. \tag{15}$$

Proof. Consider an arbitrary job $j \in J$. Let I_j be the set of all jobs which are smaller than j (or have a smaller index than job j in case there are several jobs with the same processing time). Then, the completion time of job j in schedule id is defined as follows

$$C_j(id) \;=\; p_j \;+\; \sum_{z \in I_j} p_z. \tag{16}$$

For job j, all jobs placed before j in schedule σ form two sets: J_j of medium and small jobs, and J'_j of large jobs. Then, the completion time of job j in σ is defined as follows

$$C_j(\sigma) \;=\; p_j \;+\; \sum_{k \in J_j} p_k \;+\; \sum_{\ell \in J'_j} p_\ell. \tag{17}$$

Clearly, $J_j \subseteq I_j$. Thus, the "delay" of job j is formed by large jobs ℓ in J'_j which are not in I_j.

Consider large jobs ℓ in $J'_j \setminus I_j$. These jobs are larger than job j but placed before it in schedule σ. Hence, they cannot create enough idle time on the machine to accommodate all jobs in $I_j \cup \{j\}$. (Otherwise, the algorithm can place job j on an earlier position.) Thus,

$$\gamma \cdot \sum_{\ell \in J'_j \setminus I_j} p_\ell \;<\; p_j \;+\; \sum_{z \in I_j} p_z. \tag{18}$$

Since $J_j \subseteq I_j$ and $J_j \cap J'_j = \emptyset$, we have that

$$\sum_{k \in J_j} p_k \;+\; \sum_{\ell \in J'_j \cap I_j} p_\ell \;\leq\; \sum_{z \in I_j} p_z. \tag{19}$$

Thus, combining (16), (17), (18) and (19) we prove (14). Summing up over all jobs we prove (15). □

4 Non-Existence Bounds

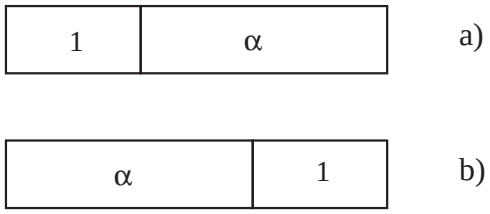

Figure 2: a) J_1 before J_2; b) J_2 before J_1

Theorem 2. *For any $\gamma > 1$, there is an instance such that no (x,y)-schedule with $1 < x < 1 + \frac{1}{\gamma}$ and $1 < y < 1 + \frac{\gamma-1}{2+\gamma}$ exists.*

Proof Sketch. We consider two jobs J_1 and J_2. Let $\omega > \alpha > 1$ be some constant. Then, we define processing times p_1 and p_2 to be 1 and α, and weights w_1 and w_2 to be 1 and ω, respectively.

Indeed, there are only two schedules:

a) J_1 before J_2, and

b) J_2 before J_1.

For an illustration see Figure 2. From the first schedule we get

$$OPT1 \;=\; 1 \;+\; (1 \;+\; \alpha) \;=\; 2 \;+\; \alpha \;\text{ and }\; W \;=\; 1 \cdot 1 + (\alpha + 1) \cdot \omega.$$

From the second schedule we get

$$S \;=\; \alpha + (1 + \alpha) = 2\alpha + 1 \;\text{ and }\; OPT2 \;=\; \alpha \cdot \omega \;+\; (\alpha + 1) \cdot 1.$$

Let $\gamma > 1$ be such that

$$S/OPT1 \;=\; \frac{2\alpha+1}{2+\alpha} \;=\; 1 \;+\; \frac{\alpha-1}{2+\alpha} \;=\; 1 \;+\; \frac{1}{\gamma}. \tag{20}$$

Then, if we want to find an (x,y)-schedule with $1 < x < 1+\frac{1}{\gamma}$ and $1 < y$, the first schedule has to be adopted. However, in this case we get

$$W/OPT2 \;=\; \frac{1+(1+\alpha)\omega}{1+\alpha\cdot\omega+\alpha} \;=\; 1 \;+\; \frac{\omega-\alpha}{1+\alpha\cdot\omega+\alpha}.$$

For $\omega \to \infty$, this bound tends to

$$1 \;+\; \frac{1}{\alpha}.$$

From (20) we can find that

$$\alpha \;=\; \frac{2+\gamma}{\gamma-1}.$$

Thus, the value of y is at least

$$1 \;+\; \frac{1}{\alpha} \;=\; 1+\frac{\gamma-1}{2+\gamma}.$$

This completes the proof. □

Conclusions

We presented the first results concerning the simultaneous approximation of two min-sum objectives, namely, the sum of completion times and the sum of weighted completion times. Many questions remain open, e.g. obtaining better lower and upper bounds for the considered problem. From another side it seems that similar algorithms, thought with different bounds, can be obtained for the cases of parallel machines, release dates and precedence constraints. Finally, a related open question concerns the simultaneous approximation of two throughput objectives, e.g., the total number of early jobs $\sum \bar{U}_j$ and the total weighted number of early jobs $\sum w_j \bar{U}_j$, where $\bar{U}_j = 1$ if job j completes before its due date, and $\bar{U}_j = 0$ otherwise.

References

[1] J. Aslam, A. Rasala, C. Stein, and N. Young. Improved bicriteria existence theorems for scheduling. In *Proceedings 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 846–847, 1999.

[2] C.H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 86–92.

[3] A. Rasala, C. Stein, E. Torng, and P. Uthaisombut. Existence theorems, lower bounds and algorithms for scheduling to meet two objectives. In *Proceedings 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 723 – 731, 2002.

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

[5] C. Stein and J. Wein. On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Operations Research Letters*, 21:115–122, 1997.