Operations Research Letters 46 (2018) 438–442

Contents lists available at ScienceDirect

Operations Research Letters

journal homepage: www.elsevier.com/locate/orl

A $(2+\epsilon)$-approximation for precedence constrained single machine scheduling with release dates and total weighted completion time objective

René Sitters [a,b,*], Liya Yang [c]

[a] VU University Amsterdam, 1081HV Amsterdam, The Netherlands
[b] Centrum Wiskunde & Informatica (CWI), 1098XG Amsterdam, The Netherlands
[c] East China University of Science and Technology, 200237 Shanghai, China

ARTICLE INFO

Article history:
Received 16 November 2017
Received in revised form 22 May 2018
Accepted 22 May 2018

Keywords:
Scheduling
Approximation algorithm
Precedence constraints
Total weighted completion time

ABSTRACT

We give a $(2+\epsilon)$-approximation algorithm for minimizing total weighted completion time on a single machine under release time and precedence constraints. This settles a recent conjecture on the approximability of this scheduling problem (Skutella, 2016).

1. Introduction

We consider the problem of minimizing the total weighted completion time on a single machine under precedence and release time constraints, denoted by $1|r_j, prec|\sum w_jC_j$ in the standard notation from [6]. An instance is given by a set of jobs $J = \{1, 2, \ldots, n\}$ and for each $j \in J$ an integer processing time $p_j \geq 0$, release time $r_j \geq 0$, and weight $w_j \geq 0$. Further, we are given a partial order $\prec$ on J representing the precedence constraints between the jobs. (The partial order is transitive, i.e., if $h \prec j$ and $j \prec k$ then $h \prec k$.) A schedule is defined by a start time $S_j \geq 0$ for each j such that no job starts before its release time, i.e., $S_j \geq r_j$, and no two jobs are processed at the same moment, i.e., for any pair j, k either $S_j \geq S_k + p_k$ or $S_k \geq S_j + p_j$, where the latter must hold in case $j \prec k$. The cost of a schedule is the weighted sum of job completion times, $\sum_j w_jC_j$, where $C_j = S_j + p_j$, and the goal is to minimize cost. We say that an algorithm is an α-approximation algorithm ($\alpha \geq 1$) if for any instance the cost of the algorithm's schedule is at most α times the optimal cost.

The special case without release time constraints ($r_j = 0$ for all jobs j) has been well studied but the computational complexity is still not completely settled. Several 2-approximation algorithms are known [3] and Bansal and Khot [2] showed that no $(2-\epsilon)$-approximation algorithm exists under the assumption that some variant of the unique games conjecture is true. It is yet unknown if this lower bound holds under the common assumption $\mathcal{P} \neq \mathcal{NP}$. For the problem with release dates, a 3-approximation algorithm was given by Schulz [9] and Hall et al. [7] using list scheduling in order of LP-values. Schulz and Skutella [10] gave an $(e+\epsilon)$-approximation algorithm by sequencing jobs in order of random α-points after solving an LP for the preemptive version. Recently, Skutella [12] improved the ratio to $\sqrt{e}/(\sqrt{e}-1) < 2.542$ and conjectured that a $(2+\epsilon)$-approximation algorithm exists. See the papers [1,3] and [12] for a more detailed overview of approximation results. Here, we give a positive answer to the conjecture by presenting a polynomial time $(2+\epsilon)$-approximation algorithm for any constant $\epsilon > 0$. Hence, our algorithm matches the lower bound for the problem without release time constraints up to an arbitrarily small factor $1+\epsilon$.

The first step of our algorithm is a refined version of a decomposition technique introduced in [11] to give a polynomial time approximation scheme for the so called Traveling Repairman Problem in the Euclidean plane and for the scheduling problem $1|prec|\sum w_jC_j$ with interval ordered precedence constraints. We show that with loss of a factor $(1+\epsilon)$ we can decompose the problem into subproblems that can be solved independently. The final schedule is obtained by placing the schedules for subproblems one after the other. The decomposition is simply done by solving an LP-relaxation and partitioning the jobs according to LP-values. The decomposition is done at random but is easy to derandomize.

* Corresponding author at: VU University Amsterdam, 1081HV Amsterdam, The Netherlands.
E-mail addresses: r.a.sitters@vu.nl (R. Sitters), liya.yang@mail.ecust.edu.cn (L. Yang).

https://doi.org/10.1016/j.orl.2018.05.007
0167-6377/© 2018 Elsevier B.V. All rights reserved.

Each subproblem has the property that all jobs are scheduled in an interval $[L, \delta L]$ where $L > 0$ and $\delta > 1$ is a constant depending on ϵ only. This property is exploited to get a $(2+\epsilon)$-approximation for subproblems. For each subproblem we work as follows. We guess the approximate start time of $O(1/\epsilon)$ jobs in an optimal schedule and use that information to strengthen the LP. Then we solve the LP and apply list scheduling in order of LP-values. Since we only guess the start time of a constant number of jobs, a polynomial number of guesses is enough and we return the best solution found.

2. List scheduling in LP-order

A common technique for minimizing total weighted completion time in scheduling is to apply list scheduling in an order that is derived from a linear program relaxation. In list scheduling, all jobs are in an ordered list and are added to the schedule one by one in an order derived from the list. In the presence of release dates, there are two intuitive versions of List Scheduling. The straightforward approach is to schedule jobs as early as possible precisely in the order of the list. Note that this may cause the machine to stay idle while jobs (later in the list) are available. Schulz [9] showed that for this version, List Scheduling in order of LP-values is a 3-approximation algorithm for $1|r_j, prec|\sum w_jC_j$. Alternatively, one may schedule at any moment that the machine is idle the job that comes earliest in the list among the available jobs. It is this latter variant that we use here.

Given a (partial) schedule, we say that the *machine is available* at time t if for any job in the schedule either $C_j \le t$ or $S_j \ge t$. We say that a *job j is available* at time t if (i) $r_j \le t$, (ii) job j was not scheduled yet, (iii) all jobs k with $k \prec j$ have been completed.

Algorithm List Scheduling (LS):

Let the jobs $J = \{1, 2, \dots, n\}$ be labeled such that $j < k$ whenever $j \prec k$. At any moment t that the machine is available, start the job with the smallest index j among the available jobs.

We assume without loss of generality that for any given instance, release dates are consistent with precedence constraints, i.e., we assume that $r_j \le r_k$ if $j \prec k$.

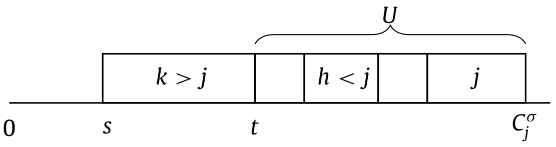
Lemma 1. *If release dates are consistent with precedence constraints, i.e., $r_j \le r_k$ if $j \prec k$, then LS has the following property: If at time t the machine is available and there is a job j with $r_j \le t$ and job j has not started yet, then LS starts some job $h \le j$ at time t.*

Proof. Let h be job with smallest index among the jobs with release time at most t and that have not been scheduled yet at time t. Clearly, $h \le j$. Further, h is available since for any $k \prec j$ we have $r_k \le r_j \le t$ and by minimality of h, job k is completed before time t. Hence, LS starts h at time t. □

The LP-formulation that we use for our problem was introduced by Queyranne [8] and is based on completion time variables only. It was later refined by Goemans [4,5] to handle release times. Although the number of constraints in the linear program is exponential, they can be separated in polynomial time by efficient submodular function minimization [4].

$$\begin{aligned}
&\min Z = \sum_{j=1}^{n} w_jC_j \\
&\text{s.t. } C_j \ge r_j + p_j && \forall j \\
&C_j \le C_k && \forall j \prec k \\
&\sum_{j\in U} p_jC_j \ge r_{\min}(U)p(U) + \frac{1}{2}p(U)^2 && \forall U \subseteq J, U \ne \emptyset.
\end{aligned}$$

In the last constraint, $p(U) = \sum_{j\in U} p_j$ and $r_{\min}(U) = \min\{r_j \mid j \in U\}$. For later use, we define $r_{\max}(U) = \max\{r_j \mid j \in U\}$ and (given a solution) define the values $C_{\min}(U)$ and $C_{\max}(U)$ in the obvious way. The constraint $C_j \ge r_j + p_j$ is not really needed here since taking $U = \{j\}$ in the last constraint implies $C_j \ge r_j + p_j/2$ which is enough for our purpose.



Lemma 2. *$p(U) \le 2C_{\max}(U) - 2r_{\min}(U)$ for any $U \subseteq J$.*

Proof. Since the lemma is obviously true for $p(U) = 0$ we may assume that $p(U) > 0$. From the last LP-constraint we have

$$p(U)C_{\max}(U) \ge \sum_{j\in U} p_jC_j \ge r_{\min}(U)p(U) + \frac{1}{2}p(U)^2,$$

for any $U \subseteq J$. Dividing both sides by $p(U)$ gives $C_{\max}(U) \ge r_{\min}(U) + \frac{1}{2}p(U)$ which is the inequality of the lemma. □

Algorithm LP+LS:

(1) Solve the linear program. Relabel such that $C_1 \le \cdots \le C_n$ and such that $j < k$ if $j \prec k$.
(2) Run list scheduling (LS) in the order $1, \dots, n$. Let C_j^σ be the completion time of job j in the final schedule σ.

Note that we denote the LP-completion time of a job j simply by C_j and denote its completion time in σ by C_j^σ. The algorithm as defined above has an unbounded approximation ratio as shown by the following example. Let $p_1 = 1, r_1 = 1, w_1 = M$, and $p_2 = M, r_2 = 0, w_2 = 0$. For large M, the optimal schedule places jobs in the order 1, 2 and has value $2M$. The algorithm however, will schedule job 2 first since it is the only available job at time 0 which gives value $(M+1)M$. If jobs are relatively small compared to their release time, to be precise if $p_j \le r_j$ for all j, then the algorithm is a 2-approximation as we show below. The proof follows easily from the next lemma that we use again in Section 4.

Lemma 3. *Let σ be a schedule returned by algorithm LP+LS. Let $j \in J$ and let t be the smallest value such that the interval $[t, C_j^\sigma]$ has no idle time and only contains jobs $h \le j$. Let U be the set of jobs processed in the interval $[t, C_j^\sigma]$. Then,*

$$C_j^\sigma \le t + 2C_j - 2r_{\min}(U). \tag{1}$$

Further, if no job completes at time t then

$$C_j^\sigma \le 2C_j. \tag{2}$$

If some job k completes at time t then

$$r_{\min}(U) > s, \tag{3}$$

where s is the start time of job k.

Proof. Note that $U \ne \emptyset$ since $j \in U$. Further, $C_j = C_{\max}(U)$ since only jobs with $h \le j$ are in U and jobs are relabeled in Step 1. Now (1) follows directly from Lemma 2.

$$C_j^\sigma = t + p(U) \le t + 2C_{max}(U) - 2r_{\min}(U) = t + 2C_j - 2r_{\min}(U).$$

If no job completes at time t then either $t = 0$ or the machine is idle just before time t. In the former case, it follows from (1) that $C_j^\sigma \leq 0 + 2C_j - 2r_{\min}(U) \leq 2C_j$. In the latter case it follows from Lemma 1 that $t = r_{\min}(U)$ which, together with (1), implies $C_j^\sigma \leq 2C_j - r_{\min}(U) \leq 2C_j$.

Now assume job k completes at time t. If $r_{\min}(U) \leq s$ then, by Lemma 1 some job $h \leq j$ must start at time s. However, $k > j$. Hence we must have $r_{\min}(U) > s$. □

Theorem 1. *If $p_j \leq r_j$ for all $j \in J$ then algorithm LP+LS is a 2-approximation.*

Proof. Apply Lemma 3 to an arbitrary job j. If no job completes at time t then by (2) $C_j^\sigma \leq 2C_j$. On the other hand, if some job k completes at time t then by (1) and (3)

$$C_j^\sigma \leq t + 2C_j - 2r_{\min}(U) < t + 2C_j - 2s.$$

Since $p_k \leq r_k$ we have $t = s + p_k \leq s + r_k \leq 2s$. Hence, $C_j^\sigma \leq 2C_j$. Now take the weighted sum over all jobs:

$$\sum_j w_j C_j^\sigma \leq 2\sum_j w_j C_j = 2Z_{LP} \leq 2\textsc{Opt},$$

where Opt is the optimal value. □

Given the theorem above, the following approach leads intuitively to a 2-approximation algorithm. Imagine an unknown optimal schedule σ' and for each job j guess if it starts before time p_j in σ' and if so, guess its precise start time. Then use this information to strengthen the LP and run algorithm LP+LS. Clearly, the running time is not polynomial in general since there can be $O(n)$ of those jobs. However, in the next section we show that, with loss of a factor $1+\epsilon$ in the approximation, one can decompose any instance I into independent sub-instance $I_1, I_2, \ldots,$ such that our guessing plus strengthening approach is polynomial for each of the sub-instances.

3. Decomposing the instance

We use the common approach of partitioning time into intervals of geometrically increasing length. However, such a partitioning gives a significant loss in the approximation ratio in general. The combination of several techniques ensures that the (expected) loss is no more than a factor $1+\epsilon$. The decomposition is done based on LP-values, i.e., the job set J is partitioned into subsets J_i where J_i is the set of jobs that have their LP-completion time C_j in the ith interval. This ensures that the schedule for J_i is only a constant factor longer than the length of the ith interval (Lemma 4). This property, together with the large factor used in the geometric grouping ($e^{3/\epsilon}$) and the randomness, ensures that the expected delay due to this partitioning is only a factor $1+\epsilon$ (Theorem 2).

For the ease of analysis we shall assume that there is no initial set U with $p(U) = 0$ since such set can be scheduled at time 0 and hence can be removed from the instance. What we get from this is that in any LP-solution, $C_j \geq 1$ for all j (using that p_j integer). Further, we assume that $\epsilon \leq 1$. The decomposition algorithm that we use here is based on a technique used for the Traveling Repairman Problem [11] and is similar to the decomposition algorithm presented in the forthcoming journal version of that paper.

Algorithm Decompose:

(1) Solve the linear program. Let $C_1, \ldots, C_n$ be the LP-values.
(2) Let $a = 3/\epsilon$ and take b uniformly at random from $[0, a]$. Let $t_i = e^{a(i-3)+b}$ for $i = 1, 2, \ldots, q+1$. Choose q large enough such that $C_{max}(J) < t_{q+1}$. Partition the jobs into $J_i = \{j | t_i \leq C_j < t_{i+1}\}, i \in \{1, 2, \ldots, q\}$.
(3) Let I_i be the scheduling instance defined by jobs J_i with the additional constraint that no job is allowed to start before time $3t_i$. For each i, run the algorithm described in Section 4 and let $\sigma_1, \ldots, \sigma_q$ be the schedules returned.
(4) Return σ which is the concatenation of $\sigma_1, \ldots, \sigma_q$.

First, let us see why the algorithm returns a feasible schedule. Step 1 is the same as in algorithm LP+LS except that relabeling is not needed here since the LP is only used to partition the instance. To see that the partition is proper note that $t_1 = e^{b-2a} \leq e^{-a} < 1 \leq C_{\min}(J)$. Further, the instances of Step 3 are well defined since we only add a lower bound of $3t_i$ to the start time of each job. Finally, note that the second LP-constraint ensures that the precedence constraints are satisfied in σ since if $k \prec j$ then $C_k \leq C_j$ and k will be scheduled before j in σ. Hence, σ is feasible if we place the partial schedules in the order $\sigma_1, \ldots, \sigma_q$ and shift schedules forward in case of overlap. However, we will show below that schedule σ_i is contained in the interval $[3t_i, 3t_{i+1}]$. That means, we can simply take the union of the σ_i and do not need to shift.

3.1. Analysis

We say that a schedule is *tight* if no job can be scheduled earlier (shifted to the left) while maintaining feasibility and without shifting any of the other jobs. Clearly, a non-tight schedule can be made tight by checking jobs one by one, starting from the left, and shift it if possible. Hence we may assume that the schedules σ_i returned by the algorithm described in Section 4 are tight.

Lemma 4. *Any tight schedule for I_i is contained in the interval $[3t_i, 3t_{i+1}]$.*

Proof. By definition, no job starts before time $3t_i$. Since the schedule is tight, the last job completes latest at time

$$\max\{3t_i, r_{\max}(J_i)\} + p(J_i).$$

Note that $r_{\max}(J_i) \leq t_{i+1}$ since $r_j \leq C_j \leq t_{i+1}$ for all jobs $j \in J_i$. Since we assume that $\epsilon \leq 1$ we have that $t_{i+1} = e^{3/\epsilon} t_i > 3t_i$. Further, by Lemma 2,

$$p(J_i) \leq 2C_{\max}(J_i) \leq 2t_{i+1}.$$

Hence, the last job completes latest at time

$$\max\{3t_i, r_{\max}(J_i)\} + p(J_i) \leq t_{i+1} + 2t_{i+1} = 3t_{i+1}. \quad \square$$

Let $\textsc{Opt}_i$ be the optimal value of instance I_i. Now, consider an optimal schedule σ^* for I and let $\textsc{Opt}|_i$ be the contribution of J_i in the optimal schedule. That means, $\textsc{Opt}|_i = \sum_{j \in J_i} w_j C_j^*$, where C_j^* is the completion time of job j in σ^*. A feasible schedule for I_i is obtained by removing the jobs not in J_i from σ^* and shifting the remaining schedule forward by at most $3t_i$. Hence, we get the following bound.

Lemma 5. $\textsc{Opt}_i \leq \textsc{Opt}|_i + 3t_i \sum_{j \in J_i} w_j$.

The value $\textsc{Opt}_i$ depends on the random variable b that defines the random partition. For job j, let $i(j)$ be such that $j \in J_{i(j)}$, that means, $t_{i(j)} \leq C_j < t_{i(j)+1}$. Note that $t_{i(j)}$ is a stochastic variable of the form $t_{i(j)} = e^{-x} C_j$, where x is uniform on $[0, a]$.

$$\mathbb{E}[t_{i(j)}] = C_j \mathbb{E}[e^{-x}] = \frac{C_j}{a} \int_{x=0}^{x=a} e^{-x} dx = \frac{C_j(1 - e^{-a})}{a} < \frac{C_j}{a}. \tag{4}$$

Lemma 6. $\mathbb{E}[\sum_i \textsc{Opt}_i] \leq (1+\epsilon)\textsc{Opt}$.

Proof. By Lemma 5,

$$\sum_i \textsc{Opt}_i \leq \sum_i \textsc{Opt}|_i + 3\sum_i \sum_{j \in J_i} w_j t_i = \textsc{Opt} + 3\sum_j w_j t_{i(j)}.$$

From (4), the expected value over b is

$$\mathbb{E}[\sum_i \textsc{Opt}_i] \leq \textsc{Opt} + 3\sum_j w_j \mathbb{E}[t_{i(j)}]$$

$$\leq \text{Opt} + \frac{3}{a}\sum_j w_jC_j \leq (1+\frac{3}{a})\text{Opt}$$
$$\leq (1+\epsilon)\text{Opt}. \quad \square$$

Remember that a schedule is called tight if no job can be shifted left (scheduled earlier) while maintaining feasibility and without shifting any of the other jobs.

Definition 1. We say that an instance of $1|r_j, prec|\sum w_jC_j$ with job set J is *δ-bounded* (where $\delta > 1$ is a constant) if there is some number $L > 0$ such that $r_{\min}(J) \geq L$ and any tight schedule completes within time δL.

Theorem 2. *For any instance I of $1|r_j, prec|\sum w_jC_j$ and constant $\epsilon > 0$ we can find δ-bounded instances $I_1, \dots, I_q$, with $\delta = e^{3/\epsilon}$, such that if $\sigma_1, \dots, \sigma_q$ are (randomized) α-approximate schedules for $I_1, \dots, I_q$ then the schedule obtained by placing the σ_i's in order $i = 1, \dots, q$ is a randomized $\alpha(1+\epsilon)$-approximate schedule for I.*

Proof. Each I_i is a bounded instance with $L = 3t_i$ and $\delta = e^{3/\epsilon}$. If each schedule σ_i is an α-approximation for instance I_i then the union is feasible (Lemma 4) and has expected value (Lemma 6) at most

$$\sum_i \mathbb{E}[\alpha\text{Opt}_i] = \alpha\sum_i \mathbb{E}[\text{Opt}_i] \leq \alpha(1+\epsilon)\text{Opt}. \quad \square$$

The theorem implies that any (randomized) polynomial time α-approximation algorithm for bounded instances yields a randomized polynomial time $\alpha(1+\epsilon)$-approximation for general instances. If the algorithm for the bounded instances is deterministic then we can easily derandomize the combined algorithm by discretizing the probability distribution for $b \in [0, a]$. We show in the next section how to get a deterministic α-approximate schedule for bounded instances with $\alpha = 2(1+\epsilon)$.

4. An algorithm for bounded instances

In this section we restrict to bounded instances as defined in Definition 1. Apart from that definition, the analysis here is independent of Section 3. Hence, let I be any bounded instance with parameters δ and L and let a constant $\epsilon > 0$ be given. We show how to get a $2(1+\epsilon)$-approximation for this instance.

The main idea of the algorithm is to guess enough information about an (unknown) optimal schedule for I such that the algorithm LP+LS of Section 2 yields a $2(1+\epsilon)$-approximate schedule. To restrict the number of guesses we first observe below that we only need to consider a nearly optimal schedule σ' in which each job j starts at a time that is a multiple of ϵp_j. Say that a job j is *early* in σ' if it starts at a time $S'_j < p_j$. We will guess the start time of each early job. The second observation is that the number of early jobs is $O(\log\delta)$ (Lemma 7) which is a constant. With these two observations, the number of guesses is polynomially bounded. For each guess, we adjust release times of jobs in correspondence with our guess and run algorithm LP+LS. Note that we do not fix the early jobs but only adjust their release times in the instance. The final solution is the best schedule over all guesses.

Restricting the optimal schedule. Let Opt be the optimal value for the bounded instance I. Consider some (unknown and tight) optimal schedule σ^* and let $C_1^* < \dots < C_n^*$ be the completion times. Assume we shift jobs one by one (starting with job 1) such that the start time of each job j is a multiple of ϵp_j. Let this schedule be σ'. Then, for any j the new completion time is

$$C'_j \leq C_j^* + \sum_{k\leq j}\epsilon p_k = C_j^* + \epsilon\sum_{k\leq j} p_k \leq (1+\epsilon)C_j^*.$$

Let Opt$'$ be the value of σ'. We have the following properties.

(i) $\text{Opt}' \leq (1+\epsilon)\text{Opt}$.
(ii) The start time S'_j of job j is a multiple of ϵp_j.
(iii) All jobs are scheduled in the interval $[L, (1+\epsilon)\delta L]$.

From now on, let σ' be our (unknown) near-optimal schedule and let Opt$'$ be its value. We will show how to get a schedule of value at most 2Opt$'$.

Algorithm Bounded:
Guess the set $A \subseteq J$ of jobs that are early in the (near) optimal schedule σ' and for each $j \in A$ guess its start time. Then, adjust the release times, $r_j \to r'_j$, and run algorithm LP+LS. Apply the steps above for all possible guesses and return the best schedule σ.

Step 1: Guessing the optimal schedule. The first step of the algorithm is to make guesses about σ'. Let S'_j be the start time of job j in σ'. Say that a job j is *early* in σ' if $S'_j < p_j$.

Lemma 7. *The number of early jobs in σ' is $O(\log\delta)$.*

Proof. If j is an early job in σ' then $C'_j > 2S'_j$. Hence, the number of early jobs is bounded by

$$\log_2\left(\frac{(1+\epsilon)\delta L}{L}\right) = \log_2((1+\epsilon)\delta) = O(\log\delta). \quad \square$$

Our algorithm guesses the set $A \subseteq J$ of early jobs in σ' and for each early job j we guess its start time S'_j in σ'. If j is early then there are at most $1/\epsilon$ possibilities to consider since $S'_j < p_j$ and S'_j is a multiple of ϵp_j. Hence, the total number of guesses is bounded by $(n/\epsilon)^{O(\log\delta)}$.

Step 2: Adjusting release times $r_j \to r'_j$. We describe this step under the assumption that our guess about σ' is correct, i.e., $A \subseteq J$ is the set of early jobs in σ' and S'_j is the start time of $j \in A$. For any early job we may adjust its release time to $r'_j = S'_j$ and for any $j \notin A$ we may define $r'_j = \max\{r_j, p_j\}$ while maintaining feasibility of σ'. Next, we may further adjust release times in correspondence with precedence constraints. The new instance I' has the following properties.

(a) Schedule σ' is feasible for I'.
(b) If $j \in A$ then $r'_j \geq S'_j$.
(c) If $j \notin A$ then $r'_j \geq \max\{r_j, p_j\}$.
(d) If $j \prec k$ then $r'_j \leq r'_k$.

Let T be the set of *open* time intervals at which an early job is processed, i.e.,

$$T = \bigcup_{j\in A}\left(S'_j, S'_j + p_j\right).$$

No job starts at a time $t \in T$ in σ' so we may increase release times further such that

(e) For any $j \in J$, we have $r'_j \notin T$.

The increase due to (e) may give a conflict with (d) causing a sequence of increases by rules (d) and (e). Clearly, this process ends after a polynomial number of iterations. Hence, we can find release times r'_j such that (a)–(e) hold.

4.1. The approximation ratio

Theorem 3. *Algorithm* Bounded *returns a $2(1+\epsilon)$-approximate solution for bounded instances.*

Proof. Assume that we guessed the information about σ' correctly. Let Z_{LP} be the LP value obtained. Then $Z_{LP} \leq \text{Opt}'$. We will show that

$C_j^\sigma \leq 2C_j$ for any job j, where C_j is the optimal LP-value for I'. Then the theorem follows by taking the weighted sum over all jobs:

$$\sum_j w_j C_j^\sigma \leq 2\sum_j w_j C_j = 2Z_{LP} \leq 2\text{OPT}' \leq 2(1+\epsilon)\text{OPT}.$$

Consider schedule σ returned by algorithm BOUNDED. We now apply Lemma 3. (See the figure.) Let j be an arbitrary job and let t be the smallest value such that the interval $[t, C_j^\sigma]$ has no idle time and only contains jobs $h \leq j$. Let U be the set of jobs processed in the interval $[t, C_j^\sigma]$. If no job completes at time t then, by (2), $C_j^\sigma \leq 2C_j$. Now assume some job k completes at time t and let s be its start time. If $t \leq 2s$ then from (1) and (3), $C_j^\sigma \leq t + 2C_j - 2r'_{\min}(U) < t + 2C_j - 2s \leq 2C_j$. Hence, assume from now that

$$t > 2s. \tag{5}$$

Then, $p_k = t - s > s \geq r'_k$ and by (c) we must have $k \in A$, i.e., k is an early job. Since $k \in A$ we have (from (b)) $S'_k \leq r'_k \leq s$ and (from (3))

$$r'_{\min}(U) > s \geq S'_k. \tag{6}$$

Also, since $k \in A$, we know from (e) that $r'_{\min}(U) \notin \,]S'_k, S'_k + p_k[$. Together with (6) and (5) we get that

$$r'_{\min}(U) \geq S'_k + p_k \geq p_k = t - s > t/2.$$

Again using (1) we conclude that

$$C_j^\sigma \leq t + 2C_j - 2r'_{\min}(U) < 2C_j. \quad \square$$

4.2. The running time

Lemma 8. *Algorithm* BOUNDED *runs in polynomial time.*

Proof. The number of guesses to consider is $(n/\epsilon)^{O(\log\delta)}$. For each guess, adjusting the release times, the LP, and list scheduling can be done in polynomial time. Hence, the total running for algorithm BOUNDED is $(n/\epsilon)^{O(\log\delta)}$. $\square$

Given the lemma above, it follows immediately that the total running time of our algorithm is polynomial. The linear program is solved once to partition an instance into $O(n)$ instances I_i. For each I_i the algorithm takes $(n/\epsilon)^{O(\log\delta)}$ time where $\log\delta = O(1/\epsilon)$. Hence, the total running time is $(n/\epsilon)^{O(1/\epsilon)}$.

Reducing the running time. We can reduce the total running time to $f(\epsilon)p(n)$ for some function f and polynomial p. To reduce the number of guesses needed, round the processing times up to powers of $1+\epsilon$. Say that job j is of *type i* if its rounded processing p'_j is $(1+\epsilon)^i$. Assume that in the near optimal schedule σ' the processing times are rounded. Note that in σ' there is at most one early job of each type. Instead of guessing all early jobs it is enough to guess which of the types do have an early job. Let $S^{(i)}$ be the smallest start time among the jobs of type i in σ'. Say that type i is early if $S^{(i)} < (1+\epsilon)^i$. Let B be the types that are (guessed) to be early. Let I' be the instance for adjusted release times r'_j defined by the following rules.

(a) Schedule σ' is feasible for I'.
(b) If $i \in B$ and j is of type i then $r'_j \geq S^{(i)}$.
(c) If $i \notin B$ and j is of type i then $r'_j \geq \max\{r_j, p'_j\}$.
(d) If $j \prec k$ then $r'_j \leq r'_k$.
(e) For any $j \in J$, we have $r'_j \notin T$.

Here, T is again the set of *open* time intervals at which an early job is processed, i.e.,

$$T = \bigcup_{i\in B} \left]S^{(i)}, S^{(i)} + (1+\epsilon)^i\right[.$$

The analysis is exactly the same except for the bound on the number of guesses. Note that if type i is early then $(1+\epsilon)^i > L$ since no job starts before time L. Further, we must have $(1+\epsilon)^i < (1+\epsilon)^2\delta L$ since no job completes after time $(1+\epsilon)^2\delta L$ in σ'. Hence, we only need to consider a range of $O(\log_{(1+\epsilon)}\delta)$ values for i. The number of guesses is bounded by $(1/\epsilon)^{O(\log_{(1+\epsilon)}\delta)} = (1/\epsilon)^{O((\log\delta)/\epsilon)}$.

Acknowledgments

This research was done while visiting the Vrije Universtiteit Amsterdam and was supported in part by the NSFC 11371137, 71431004, and by the China Scholarship Council under Grant No. 201606740022.

References

[1] C. Ambühl, M. Mastrolilli, N. Mutsanas, O. Svensson, On the approximability of single-machine scheduling with precedence constraints, Math. Oper. Res. 36 (2011) 653–669.
[2] N. Bansal, S. Khot, Optimal long code test with one free bit, in: Proc. of the 50th Annual IEEE Symposium on Foundations of Computer Science, 2009 pp. 453–462.
[3] C. Chekuri, S. Khanna, Approximation algorithms for minimizing average weighted completion time, in: J. Leung (Ed.), Handbook of Scheduling: Algorithms, Models, and Performance Analysis, CRC Press, 2004.
[4] M.X. Goemans, A supermodular relaxation for scheduling with release dates, in: W.H. Cunningham, S.T. McCormick, M. Queyranne (Eds.), Proc. 5th Int. Conf. Integer Programming and Combinatorial Optimization, in: Lecture Notes in Computer Science, vol. 1084, Springer, 1996, pp. 288–300.
[5] M.X. Goemans, Improved approximation algorithms for scheduling with release dates, in: Proc. 8th Symp. on Discrete Algorithms, New Orleans, Louisiana, United States, 1997, pp. 591–598.
[6] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5 (1979) 287–326.
[7] L.A. Hall, A.S. Schulz, D.B. Shmoys, J. Wein, Scheduling to minimize average completion time: Off-line and on-line approximation algorithms, Math. Oper. Res. 22 (1997) 513–544.
[8] M. Queyranne, Structure of a simple scheduling polyhedron, Math. Program. 58 (1993) 263–285.
[9] A.S. Schulz, Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds, in: W.H. Cunningham, S.T. McCormick, M. Queyranne (Eds.), Proc. 5th Int. Conf. Integer Programming and Combinatorial Optimization, in: Lecture Notes in Computer Science, vol. 1084, 1996, pp. 301–315.
[10] A.S. Schulz, M. Skutella, Random-based scheduling: New approximations and LP lower bounds, in: J. Rolim (Ed.), Randomization and Approximation Techniques in Computer Science, in: Lecture Notes in Computer Science, vol. 1269, 1997, pp. 119–133.
[11] R.A. Sitters, Polynomial time approximation schemes for the traveling repairman and other minimum latency problems, in: Proc. 25th Symp. on Discrete Algorithms, 2014, pp. 604–616.
[12] M. Skutella, A 2.542-approximation for precedence constrained single machine scheduling with release dates and total weighted completion time objective, Oper. Res. Lett. 44 (2016) 676–679.