Combinatorial Algorithms for Minimizing the Weighted Sum of Completion Times on a Single Machine

James M. Davis[1]

Department of Operations Research, Cornell University, Ithaca, NY 14850.

Rajiv Gandhi

Department of Computer Science, Rutgers University-Camden, Camden, NJ 08102.

Vijay Kothari

Department of Computer Science, Dartmouth College, Hanover, NH 03755.

Abstract

We study the problem of minimizing the weighted sum of completion times of jobs with release dates on a single machine with the aim of shedding new light on "the simplest [linear program] relaxation" [17]. Specifically, we analyze a 3-competitive online algorithm [16], using dual-fitting. In the offline setting, we develop a primal-dual algorithm with approximation guarantee $1 + \sqrt{2}$. The latter implies that the cost of the optimal schedule is within a factor of $1 + \sqrt{2}$ of the cost of the optimal LP solution.

Keywords: scheduling, algorithms, online, approximation, primal-dual, dual-fitting

1. Introduction

We consider the problem of minimizing the weighted sum of completion times on a single machine with release dates, denoted by $1|r_j|\sum_j w_j C_j$ [9]. In this problem we are given a set of n jobs $J = \{1, 2, \ldots, n\}$, each with a processing time $p_j > 0$, weight $w_j \geq 0$ and release date $r_j \geq 0$. The objective is to schedule these jobs non-preemptively on a single machine to minimize $\sum_j w_j C_j$, where C_j denotes the completion time of job j in the schedule. This problem is NP-hard, even when each job has a unit weight [13]. When all jobs have the same release dates the problem is solved optimally by using Smith's rule [21].

This problem has been studied extensively in the offline as well as online settings. In the offline case, the algorithm has complete knowledge of all jobs when constructing the schedule, however, in the online setting, we gain knowledge of a job on its release date and for each time t we must construct the schedule until time t without any knowledge of jobs that are released afterwards. For the offline problem, polynomial time approximation schemes have been developed [1]. There are also several linear programming based approximations [3, 4, 7, 10, 18, 19, 20]. The linear programming based techniques derive primarily from three different LP formulations (discussed in detail in [17]): the completion time LP (LP1), the completion time LP with shifted parallel inequalities (LP2), and the preemptive time indexed LP (LP3). Most algorithms use LP2 and LP3 [3, 4, 7, 19, 20]. Goemans et al. [7] study these two LP formulations in detail, show their equivalence and present a LP-rounding algorithm with an approximation guarantee of 1.6853. They also give an online algorithm with a competitive ratio of $(1 + \sqrt{2})$. LP1 is not so well studied. Schulz [18] and Hall et al. [10] use LP1 to derive a 3-approximation for the problem using LP-rounding.

Our work focuses on understanding LP1. There are several online algorithms [2, 11, 14, 16, 22] that are based on the idea of postponing release dates of jobs so that a job with a large processing time and a small release date has to wait before it can actually start processing. In particular, Megow and Schulz [16] give a deterministic online algorithm (that is an extension of earlier work [2, 11, 14]) that is 3-competitive when restricted to a single machine. We give an alternate proof of the result using the dual-fitting method. We believe that this analysis may find independent use in analyzing algorithms using such linear programs [5, 6, 12]. The second algorithm is a primal-dual algorithm that yields an approximation guarantee of $(1 + \sqrt{2})$. Similar algorithms have been used earlier [6, 8, 15]. Our result shows that the optimal solution to the problem is within a factor $(1 + \sqrt{2})$ from the optimal solution to LP1. Schulz [18] and Hall et al. [10] had shown that the optimal solution was within a factor of 3 from

Email addresses: jmd388@cornell.edu (James M. Davis), rajivg@camden.rutgers.edu (Rajiv Gandhi), Vijay.H.Kothari@dartmouth.edu (Vijay Kothari)

[1]Corresponding author.

the optimal solution to LP1. We show that our analysis is tight and we also give a gap example that shows that using only an optimal solution to LP1 as a lower bound, we cannot obtain better than a 2-approximate solution to the problem.

Note that none of our results improve the best known performance bounds; for both problems we consider, there are algorithms with strictly better performance guarantees. Our work focuses on using what appears be a relatively weak linear programming relaxation in deriving the performance guarantees of the algorithms.

2. Linear Program Formulation

Several linear programming relaxations for the problem are well known [17]. The linear programming relaxation we use was first studied extensively by Schulz [18].

For a job j let C_j represent its completion time. For any set $S \subseteq J$ let $p(S) = \sum_{j \in S} p_j$ and $p^2(S) = \sum_{j \in S} p_j^2$. The completion time linear programming formulation is given by

$$\begin{aligned} \min \quad & \sum_{j \in J} w_j C_j & \\ \text{subject to} \quad & C_j \geq r_j + p_j, & \forall j \in J \\ & \sum_{j \in S} p_j C_j \geq \frac{p(S)^2 + p^2(S)}{2}, & \forall S \subseteq J \\ & C_j \geq 0, & \forall j \in J \end{aligned}$$

The justification for the second constraint is as follows. By the problem definition no two jobs can be scheduled at the same time. Consider any schedule for the jobs in $S \subseteq J$. Assume w.l.o.g. that the jobs are ordered by their completion time. If we set $r_j = 0$ for all jobs, then there is no time the machine is not processing a job. In this case we get that $C_j = \sum_{k=1}^{j} p_k$ and using algebra

$$\sum_{j=1}^{|S|} p_j C_j \geq \sum_{j=1}^{|S|} p_j \sum_{k=1}^{j} p_k = \sum_{j=1}^{|S|} \sum_{k=1}^{j} p_j p_k = \frac{p(S)^2 + p^2(S)}{2}.$$

Combining this with the fact that $\sum_{j=1}^{|S|} p_j C_j$ can only be greater when there are non-zero release times gives us the constraint.

The dual linear program is given by

$$\begin{aligned} \max \quad & \sum_{j \in J} \alpha_j (r_j + p_j) + \sum_{S \subseteq J} \beta_S \Big(\frac{p(S)^2 + p^2(S)}{2} \Big) & \\ \text{subject to} \quad & \alpha_j + p_j \sum_{S \,:\, j \in S} \beta_S \leq w_j, & \forall j \in J \\ & \alpha_j \geq 0, & \forall j \in J \\ & \beta_S \geq 0, & \forall S \subseteq J \end{aligned}$$

Notice that there is a dual variable α_j for every job j, a constraint for every job j, and a dual variable β_S for every subset of jobs S. The cost of any feasible dual solution is a lower bound on OPT, the cost of an optimal solution.

3. Online Algorithm

For completeness, we describe below the online algorithm in [16], when restricted to a single machine. The first step in the algorithm below is to sort the jobs by non-increasing order of weight over processing time so that the set $J' = \{1, 2, \ldots, n\}$ of jobs satisfies $\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \ldots \frac{w_n}{p_n}$. The main idea behind the algorithm is that after a job is released it is forced to wait from time r_j to $r_j + p_j$, after which it is eligible to be scheduled. When a job has finished waiting we say that it is available. When the machine is free (no job is being processed), among the available jobs we select the job with the smallest index in J' to be scheduled.

Algorithm 1 Online Algorithm

```
J' ← list of jobs sorted by non-increasing w_j/p_j
Q ← ∅
t ← 0
while J' ≠ ∅ do
  if t = r_j + p_j for some j ∈ J' then
    Q ← Q ∪ {j}
    J' ← J' − j
  end if
  if machine is not processing a job then
    if Q ≠ ∅ then
      j' ← job in Q that appears earliest in J'
      schedule j'
      Q ← Q − {j'}
    end if
  end if
end while
```

Although this algorithm runs in pseudopolynomial time it can easily be made to run in $O(n \log n)$ time. We simply need to introduce a variable indicating when the machine will finish processing the current job, say s, and increment time steps by $\min\{\min_{j \in J'} r_j + p_j, s\}$.

Note that the algorithm is an online algorithm because at any time t we only consider the jobs with release times less than t to be scheduled. Also, at any time t the schedule before t cannot be altered. What remains to be analyzed is the performance guarantee achieved by this algorithm.

3.1. Analysis

To analyze the performance guarantee we first construct a dual infeasible solution. We let S_j denote the first j jobs, $\{1, 2 \ldots j\}$, after the jobs have been sorted by non-increasing $\frac{w_j}{p_j}$ value. For convenience we let β_j denote β_{S_j}.

$$\begin{aligned} & \alpha_j = w_j, & \forall j \in J \\ & \beta_j = \frac{w_j}{p_j} - \sum_{k > j} \beta_k, & \text{for } j = n \text{ down to } 1 \\ & \beta_S = 0, & \forall S \mid \forall i \in J, S \neq S_i \end{aligned}$$

This infeasible solution is not a lower bound on OPT. We can, however, scale the values of the variables in the infeasible solution to create a feasible solution. This is formalized in Lemma 3.1.

Lemma 3.1.
$\frac{2}{3}\sum_{j\in J}\alpha_j(r_j+p_j)+\frac{1}{3}\sum_{k\in J}\beta_k\Big(\frac{p(S_k)^2+p^2(S_k)}{2}\Big)\le OPT$

Proof. We can verify that in the infeasible solution above, the nonzero α_j variables constitute a feasible dual solution. Similarly the set of nonzero β_S variables constitute a feasible solution. Since any convex combination of two feasible solutions yields a feasible solution, by scaling the α_j variables by a factor of $\frac{2}{3}$ and the β_S variables by a factor of $\frac{1}{3}$ we obtain a new feasible solution. Since any dual feasible solution is a lower bound on OPT the claim follows. □

We will need two key lemmas to relate the cost of our solution to the dual feasible solution described in Lemma 3.1.

Lemma 3.2. *For any job j we have that $w_j=\alpha_j$ and $\frac{w_j}{p_j}=\sum_{i\ge j}\beta_i$.*

Proof. This follows directly from the construction of the dual infeasible solution. □

A more general version of the following lemma for parallel machines is proved in [16].

Lemma 3.3. *The completion time, C_j, of any job j in our schedule satisfies $C_j\le 2(r_j+p_j)+p(S_j)$.*

Proof. Note that job j is eligible for processing at time r_j+p_j. Let's first assume that no other job is being processed at r_j+p_j. After j has completed its idle time, the most that j will have to wait before it begins processing is the amount of time it takes for j to be emptied from Q, which is at most $p(S_j)$. This implies that $C_j\le r_j+p_j+p(S_j)$.

If at time r_j+p_j another job k is processing, then job k must have already completed its idling period, so that $r_k+p_k\le r_j+p_j$. Therefore, $p_k\le r_j+p_j$ so that k will finish processing by $r_j+p_j+p_k\le r_j+p_j+(r_j+p_j)=2(r_j+p_j)$. Thus, $C_j\le 2(r_j+p_j)+p(S_j)$ as desired. □

We can now bound the cost of our solution.

Theorem 3.4. *The online algorithm yields a 3-approximate solution.*

Proof. We will use Lemma 3.2 and Lemma 3.3 to rewrite the cost of our solution in terms of α_j, β_j, p_j and r_j.

$$\begin{aligned}
Cost &= \sum_{j\in J} C_j w_j \\
&\le 2\sum_{j\in J} w_j(r_j+p_j)+\sum_{j\in J} p(S_j)\Big(\frac{w_j}{p_j}\Big)p_j && \text{(Lemma 3.3)}\\
&= 2\sum_{j\in J}\alpha_j(r_j+p_j)+\sum_{j\in J}p_j\sum_{k\ge j}\beta_k p(S_j) && \text{(Lemma 3.2)}\\
&= 2\sum_{j\in J}\alpha_j(r_j+p_j)+\sum_{k\in J}\beta_k\Big(\frac{p(S_k)^2+p^2(S_k)}{2}\Big)
\end{aligned}$$

Finally, we can use Lemma 3.1 to bound the solution cost.

$$\begin{aligned}
Cost &\le 2\sum_{j\in J}\alpha_j(r_j+p_j)+\sum_{k\in J}\beta_k\Big(\frac{p(S_k)^2+p^2(S_k)}{2}\Big)\\
&= 3\Big(\frac{2}{3}\sum_{j\in J}\alpha_j(r_j+p_j)+\frac{1}{3}\sum_{k\in J}\beta_k\Big(\frac{p(S_k)^2+p^2(S_k)}{2}\Big)\Big)\\
&\le 3\cdot OPT
\end{aligned}$$

□

4. Primal-Dual Algorithm

The primal-dual algorithm is inspired by the work of Gandhi and Mestre [6]. In the algorithm a feasible schedule is built iteratively from "right to left", i.e., it first decides which job to process last, then second-to-last, and so on. Consider a particular iteration. Let J' be the set of jobs that aren't scheduled at the beginning of this iteration and let j be the job with the largest release time. In any iteration we need to decide whether to increase an α_j dual variable or a $\beta_{J'}$ variable. We will use the dual LP as a guide for deciding which variable to increase in any iteration. If r_j is very large we make large gains in the dual objective function value by raising α_j. However, if $p(J')$ is large, $(p(J')^2+p^2(J'))/2$ is large and we make large gains in the objective value by raising $\beta_{J'}$. Let κ be some constant that will be optimized later. If $r_j>\kappa\cdot p(J')$ we raise the dual variable α_j until the dual constraint for j becomes tight. We then schedule j to be processed as early as possible and before every previously scheduled job.

If $r_j\le\kappa\cdot p(J')$ we raise the dual variable $\beta_{J'}$ until one of the constraints becomes tight for some job $j'\in J'$. Job j' is scheduled to be processed as early as possible and before every previously scheduled job.

This algorithm can be implemented in $O(n\log n)$ time by maintaining two sorted lists of jobs: one sorted by non-increasing r_j value and the other sorted by non-increasing $\frac{w_j}{p_j}$ value. We then observe that when $r_j>\kappa\cdot p(J)$ the job with highest r_j value is removed and when $r_j\le\kappa\cdot p(J)$ the job with lowest $\frac{w_j}{p_j}$ value is removed (observe that at

Algorithm 2 Primal-Dual

$J' \leftarrow J$
while $J' \neq \emptyset$ **do**
  $j \leftarrow$ job with largest r_j value
  if $r_j > \kappa \cdot p(J')$ **then**
    $\alpha_j \leftarrow w_j - p_j \sum_{S:j\in S} \beta_S$
    $J' \leftarrow J' - j$
  else if $r_j \leq \kappa \cdot p(J')$ **then**
    $j' \leftarrow \arg\min_j \{\frac{w_j}{p_j} - \sum_{S:j\in S} \beta_S\}$
    $\beta_{J'} \leftarrow \frac{w_{j'}}{p_{j'}} - \sum_{S:j'\in S} \beta_S$
    $J' \leftarrow J' - j'$
  end if
  schedule the jobs in the reverse order that they were removed from J'
end while

any point in time the second term, $\sum_{S:j\in S} \beta_S$ of arg min in the pseudo-code is the same for all unscheduled jobs at that time).

4.1. Analysis for Primal-Dual Algorithm

At any time during the algorithm the nonzero variables constitute a feasible dual solution. Assume w.l.o.g. that the jobs in $J = \{1, 2, \ldots, n\}$ are indexed by their order in the schedule. That is, if j and k are jobs with $j < k$ then j is scheduled before k. Let S_j be the set of jobs $\{1, 2, \ldots, j\}$. We let β_j denote β_{S_j} for convenience.

Lemma 4.1. *The following are properties of our algorithm.*

(a) Every nonzero β_S variable can be written as β_j for some job j.

(b) For every set S_j that has a nonzero β_j variable, if $i \leq j$ then $r_i \leq \kappa \cdot p(S_j)$.

(c) For every job j that has a nonzero α_j variable, $r_j > \kappa \cdot p(S_j)$.

(d) For every job j that has a nonzero α_j variable, if $i \leq j$ then $r_i \leq r_j$.

Each of the above observations can easily be verified. We will now prove two lemmas that will help us relate the cost of our solution to that of the constructed dual feasible solution. Lemma 4.2 is well known [23]; we prove it below for sake of completeness.

Lemma 4.2. *For every job j, $C_j \leq \max_{i\leq j}\{r_i\} + p(S_j)$.*

Proof. Let $r = \max_{i\leq j}\{r_i\}$. After time r, all jobs in S_j are released. Hence, after time r job j will take at most $p(S_j)$ additional time to complete. The lemma follows. □

Lemma 4.3. *For every job j, $w_j = \alpha_j + p_j \sum_{k\geq j} \beta_k$.*

Proof. To prove the lemma we simply take note of the fact that a job j isn't removed from J' until the constraint for j becomes tight. Since all jobs are removed from J' all constraints are tight. □

Theorem 4.4. *The algorithm above gives a $(1+\sqrt{2})$-approximation algorithm for $1|r_j|\sum_{j\in J} w_j C_j$.*

Proof. We use Lemma 4.3 to rewrite the cost of our solution in terms of the dual variables.

$$\begin{aligned} Cost = \sum_{j\in J} w_j C_j &= \sum_{j\in J} (\alpha_j + p_j \sum_{k\geq j} \beta_k) C_j \\ &= \sum_{j\in J} \alpha_j C_j + \sum_{j\in J} p_j \sum_{k\geq j} \beta_k C_j \qquad (1) \end{aligned}$$

We will first bound $\sum_{j\in J} \alpha_j C_j$.

$$\begin{aligned} \sum_{j\in J} \alpha_j C_j &\leq \sum_{j\in J} \alpha_j (\max_{i\leq j}\{r_i\} + p(S_j)) \qquad \text{(Lemma 4.2)} \\ &= \sum_{j\in J} \alpha_j (r_j + p(S_j)) \qquad \text{((d) of Lemma 4.1)} \\ &< (1 + \frac{1}{\kappa}) \sum_{j\in J} \alpha_j (r_j + p_j) \\ &\qquad \text{((c) of Lemma 4.1)} \quad (2) \end{aligned}$$

Now we bound $\sum_{j\in J} p_j \sum_{k\geq j} \beta_k C_j$.

$$\begin{aligned} \sum_{j\in J} p_j \sum_{k\geq j} \beta_k C_j &\leq \sum_{j\in J} p_j \sum_{k\geq j} \beta_k \big(\max_{i\leq j}\{r_i\} + p(S_j)\big) \\ &\qquad \text{(Lemma 4.2)} \\ &\leq \sum_{k\in J} \beta_k \sum_{j\leq k} p_j \big(\max_{i\leq k}\{r_i\} + p(S_j)\big) \\ &\leq \kappa \sum_{k\in J} \beta_k p(S_k) \sum_{j\leq k} p_j + \sum_{k\in J} \beta_k \sum_{j\leq k} p_j p(S_j) \\ &\qquad \text{((b) of Lemma 4.1)} \\ &= \kappa \sum_{k\in J} \beta_k p(S_k)^2 + \sum_{k\in J} \beta_k \Big(\frac{p(S_k)^2 + p^2(S_k)}{2}\Big) \\ &\leq (2\kappa + 1) \sum_{k\in J} \beta_k \Big(\frac{p(S_k)^2 + p^2(S_k)}{2}\Big) \qquad (3) \end{aligned}$$

Combining (1), (2) and (3) we get

$$Cost \leq (1+\frac{1}{\kappa}) \sum_{j\in J} \alpha_j (r_j + p_j) + (2\kappa+1) \sum_{k\in J} \beta_k \Big(\frac{p(S_k)^2 + p^2(S_k)}{2}\Big)$$

To get the best approximation guarantee we optimize κ. κ will be optimal when $1 + \frac{1}{\kappa} = 2\kappa + 1$, which gives us that $\kappa = \frac{\sqrt{2}}{2}$. This lets us derive the approximation guarantee.

$$\begin{aligned} Cost &\leq (1+\sqrt{2}) \Big(\sum_{j\in J} \alpha_j (r_j + p_j) + \sum_{k\in J} \beta_k \Big(\frac{p(S_k)^2 + p^2(S_k)}{2}\Big)\Big) \\ &\leq (1+\sqrt{2}) \cdot OPT \end{aligned}$$

□

4.2. Tight Example

The following family of instances show that the above analysis is tight. Suppose there are two jobs, job 1 and job 2. Job 1 is released at time $t-1$, has processing time $p-1$, and unit weight. Job 2 is released at time t, has unit processing time, and a sufficiently large weight $W >> p$. We let $t = \lceil p/\sqrt{2} \rceil$. Observe that our algorithm will process job 1 before processing job 2; thus the cost of our solution is given by

$$t + p - 2 + W(t + p - 1) = (W + 1)(t + p - 1) - 1$$

The optimal schedule will process job 2 before processing job 1 and its cost is given by

$$W(t + 1) + t + p = (W + 1)(t + 1) + p - 1$$

For sufficiently large values of W ($W >> p$ and $W >> t$), the ratio of our cost to the optimal approaches

$$\frac{t + p - 1}{t + 1} < \frac{t(1 + \sqrt{2}) - 1}{t + 1} \approx 1 + \sqrt{2}$$

Note that in the above instance, replacing job 1 by $p-1$ jobs, each having unit processing time and unit weight would also serve as a tight example.

4.3. Gap Example

We now show that using only an optimal solution to LP1 as a lower bound we cannot obtain better than a 2-approximation for the single machine scheduling problem.

Consider k jobs, each of unit weight and having unit processing time. All jobs are released at time t. Let $k = 2t + 1$. In an optimal LP solution, the completion time of each job is $t + 1$. Thus, the cost of the optimal solution to LP1 is $k(t+1)$. An optimal solution to the instance would schedule the jobs one after the other starting at time t. The cost of the optimal solution is given by

$$\sum_{i=t+1}^{t+k} i = k\left(t + \frac{k+1}{2}\right) = k\left(t + \frac{2t+2}{2}\right) = k(2t + 1)$$

Note that for sufficiently large values of t, the above cost approaches twice the cost of an optimal solution to LP1.

Acknowledgements: This research was supported by NSF awards 0830569 and 1050968. Part of this work was done when James Davis was an undergraduate student at Rutgers University-Camden and Vijay Kothari was visiting Rutgers University-Camden as a post-baccalaureate student. We are very grateful to an anonymous referee for a very thorough report and for suggesting various improvements in the write-up of this paper.

References:

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, et al., Approximation schemes for minimizing average weighted completion time with release dates, in: Proceedings of the 40th annual Symposium on Foundations of Computer Science., IEEE, 1999, pp. 32–43.

[2] E. Anderson, C. Potts, On-line scheduling of a single machine to minimize total weighted completion time, in: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, 2002, pp. 548–557.

[3] S. Chakrabarti, C. Phillips, A. Schulz, D. Shmoys, C. Stein, J. Wein, Improved scheduling algorithms for minsum criteria, Automata, Languages and Programming (1996) 646–657.

[4] C. Chekuri, R. Motwani, B. Natarajan, C. Stein, Approximation techniques for average completion time scheduling, SIAM Journal on Computing 31 (1) (2001) 146–166.

[5] R. Gandhi, M. M. Halldórsson, G. Kortsarz, H. Shachnai, Improved results for data migration and open shop scheduling, ACM Transactions on Algorithms (TALG) 2 (1) (2006) 116–129.

[6] R. Gandhi, J. Mestre, Combinatorial algorithms for data migration to minimize average completion time, Algorithmica 54 (1) (2009) 54–71.

[7] M. Goemans, M. Queyranne, A. Schulz, M. Skutella, Y. Wang, Single machine scheduling with release dates, SIAM Journal on Discrete Mathematics 15 (2) (2002) 165–192.

[8] M. Goemans, D. Williamson, Two-dimensional gantt charts and a scheduling algorithm of lawler, SIAM Journal on Discrete Mathematics 13 (3) (2000) 281–294.

[9] R. Graham, E. Lawler, J. Lenstra, A. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of discrete mathematics 5 (2) (1979) 287–326.

[10] L. Hall, A. Schulz, D. Shmoys, J. Wein, Scheduling to minimize average completion time: Off-line and on-line approximation algorithms, Mathematics of Operations Research 22 (3) (1997) 513–544.

[11] J. Hoogeveen, A. Vestjens, Optimal on-line algorithms for single-machine scheduling, Integer Programming and Combinatorial Optimization (1996) 404–414.

[12] Y. Kim, Data migration to minimize the average completion time, in: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2003, pp. 97–98.

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

[14] X. Lu, R. Sitters, L. Stougie, A class of on-line scheduling algorithms to minimize total completion time, Operations Research Letters 31 (3) (2003) 232–236.

[15] M. Mastrolilli, M. Queyranne, A. Schulz, O. Svensson, N. Uhan, Minimizing the sum of weighted completion times in a concurrent open shop, Operations Research Letters 38 (5) (2010) 390–395.

[16] N. Megow, A. Schulz, On-line scheduling to minimize average completion time revisited, Operations Research Letters 32 (5) (2004) 485–490.

[17] M. Queyranne, A. Schulz, Polyhedral approaches to machine scheduling, TU Berlin, Fachbereich 3, Preprint 408/1994.

[18] A. Schulz, Scheduling to minimize total weighted completion time: Performance guarantees of lp-based heuristics and lower bounds, Integer Programming and Combinatorial Optimization (1996) 301–315.

[19] A. Schulz, M. Skutella, Scheduling-lps bear probabilities randomized approximations for min-sum criteria, in: Proceedings of the fifth annual European Symposium on Algorithms (ESA'97), Springer, 1997, pp. 416–429.

[20] A. Schulz, M. Skutella, The power of α-points in preemptive single machine scheduling, Journal of Scheduling 5 (2) (2002) 121–133.

[21] W. Smith, Various optimizers for single-stage production, Naval Research Logistics Quarterly 3 (1-2) (1956) 59–66.

[22] A. Vestjens, On-line machine scheduling, Ph.D. thesis, Eindhoven University of Technology (1997).

[23] D. Williamson, D. Shmoys, The Design of Approximation Algorithms, Cambridge University Press, 2011.