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

James M. Davis¹, Rajiv Gandhi², and Vijay Kothari

¹ Department of Computer Science, Rutgers University-Camden, Camden, NJ 08102.
  jamesdav@camden.rutgers.edu

² Department of Computer Science, Rutgers University-Camden, Camden, NJ 08102.
  rajivg@camden.rutgers.edu

Abstract. In this paper we study the problem of minimizing the weighted sum of completion times of jobs with release dates on a single machine. We develop two algorithms that rely on “the simplest linear program relaxation” [8]. For the first algorithm we consider the online setting where we gain knowledge of a job on its release date and produce a schedule as the machine processes jobs. We develop an online dual fitting algorithm with an approximation guarantee of 3. This is the first online algorithm to use this LP as a lower bound. For the second algorithm we work in the off-line setting and develop a primal-dual algorithm with an approximation guarantee of 2.42. This algorithm provides the current best upper bound on the integrality gap of this simple LP formulation.

1 Introduction

Scheduling problems involve scheduling jobs on machines to optimize some objective function. Many scheduling problems have been studied in the literature, each defined by different objective functions and different constraints on admissible schedules. For an overview of these problems we refer the reader to the surveys of Queyranne and Schulz [8], Graham et al. [1] and Chekuri and Khanna [16].

In this paper we study the problem of minimizing the weighted sum of completion times on a single machine with release dates, denoted by 1|rj|∑j wjCj [1]. In this problem we are given a set of n jobs J = {1, 2, . . . , n}, each with a processing time pj > 0, weight wj ≥ 0 and release date rj ≥ 0. Our aim is to schedule these jobs non-preemptively on a single machine to minimize ∑j wjCj, where Cj denotes the completion time of job j in the schedule. This problem is NP-hard, even if the weight of every job is 1 [2]. When all release dates are 0 the problem is solved optimally by using Smith’s rule.

A significant amount of work has been done on this problem. Polynomial time approximation schemes have been discovered [3, 16]. There are also several linear programming based approximations [14, 13, 17–19, 9, 12]. The linear programming based techniques derive primarily from three different LP formulations (discussed in detail in [8]): 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 [14, 13, 17–19]. Goemans et al. [14] study these two LP formulations in detail, show their equivalence and present a LP-rounding algorithm with an approximation guarantee of 1.6853. LP1 is not so well studied. Schulz [12] studies LP1 and derives a constant factor approximation using it. Hall et al. [9] improves on this result and derives a 3 approximation algorithm using LP1, the current best guarantee for the problem using this LP. Like most other algorithms, the algorithms of Schulz and Hall et al. use the technique of LP rounding.

* Research supported by Rutgers University Research Council Grant and by NSF award CCF-0830569
Our work focuses on understanding LP1; we provide two combinatorial algorithms whose approximation guarantees give an upper bound on LP1’s integrality gap. The first algorithm is an online dual-fitting algorithm that achieves an approximation guarantee of 3. 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 [9]. To the best of our knowledge this is the first online algorithm that uses LP1; hence it is the first online algorithm to give an upper bound on the integrality gap of LP1. The second algorithm is a primal-dual algorithm that yields an approximation guarantee of \( 1 + \sqrt{2} \approx 2.42 \), improving on the result of Hall et al. [9]. The second algorithm gives the current best upper bound on the integrality gap of LP1. Both algorithms are simple to implement and run in \( O(n \log(n)) \) time.

2 Linear Program Formulation

Several linear programming relaxations for the problem are well known [8]. The linear programming relaxation we use was first studied extensively by Schulz [12].

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

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

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 \( C_j = \sum_{k=1}^{|S|} p_k \) and using algebra

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

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

\[
\begin{align*}
\max & \sum_{j \in J} \alpha_j (r_j + p_j) + \sum_{S \subseteq J} \beta_S \left( \frac{p(S)^2 + p^2(S)}{2} \right) \\
\text{subject to} & \quad \alpha_j + p_j \sum_{S \mid j \in S} \beta_S \leq w_j, & \forall j \in J \\
& \quad \alpha_j \geq 0, & \forall j \in J \\
& \quad \beta_S \geq 0, & \forall S \subseteq J
\end{align*}
\]
Notice that there is a dual variable $\alpha_j$ for every job $j$, a constraint for every job $j$, and a dual variable $\beta_S$ for every subset of jobs $S$. The cost of any feasible dual solution is a lower bound on $OPT$.

3 Dual Fitting Algorithm

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 $\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \ldots \geq \frac{w_n}{p_n}$. 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 Dual Fitting Algorithm

1. $J' \leftarrow$ list of jobs sorted by non-increasing $\frac{w_j}{p_j}$
2. $Q \leftarrow \emptyset$
3. $t \leftarrow 0$
4. while $J' \neq \emptyset$ do
5.   if $t = r_j + p_j$ for some $j \in J'$ then
6.     $Q \leftarrow Q \cup \{j\}$
7.     $J' \leftarrow J' - j$
8.   end if
9.   if machine is not processing a job then
10.      if $Q \neq \emptyset$ then
11.         $j' \leftarrow$ job in $Q$ that appears earliest in $J'$
12.         schedule $j'$
13.         $Q \leftarrow Q - \{j'\}$
14.      end if
15.   end if
16. 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 $\min\{\min_{j \in J'} r_j + p_j, s\}$.

Notice that this dual fitting algorithm is an online algorithm. At any time $t$ we have only considered 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 for Dual Fitting Algorithm

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
Lemma 1. For any job described in Lemma 1, the completion time, \( C_j \), of any job \( j \) in our schedule satisfies \( C_j \leq 2(r_j + p_j) + p(S_j) \).

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 \( C_j \leq r_j + p_j + p(S_j) \).

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

We can now bound the cost of our solution.

Theorem 1. The dual fitting algorithm is a 3-approximation.

Proof. We will use use Lemma 2 and Lemma 3 to rewrite the cost of our solution in terms of \( \alpha_j \), \( \beta_j \), \( p_j \) and \( r_j \).

\[
\text{Cost} = \sum_{j \in J} C_j w_j = 2 \sum_{j \in J} w_j (r_j + p_j) + \sum_{j \in J} p(S_j) \left( \frac{w_j}{p_j} \right) p_j
\]

\[
= 2 \sum_{j \in J} \alpha_j (r_j + p_j) + \sum_{j \in J} p_j \sum_{k \geq j} \beta_k p(S_j)
\]

\[
= 2 \sum_{j \in J} \alpha_j (r_j + p_j) + \sum_{k \in J} \beta_k \left( \frac{p(S_k)^2 + p^2(S_k)}{2} \right)
\]
Finally, we can use Lemma 1 to bound the solution cost.

\[
\text{Cost} \leq 2\sum_{j \in J} \alpha_j (r_j + p_j) + \sum_{k \in J} \beta_k \left( \frac{p(S_k)^2 + p^2(S_k)}{2} \right) \\
= 3 \left( 2\sum_{j \in J} \alpha_j (r_j + p_j) + \frac{1}{3} \sum_{k \in J} \beta_k \left( \frac{p(S_k)^2 + p^2(S_k)}{2} \right) \right) \\
\leq 3 \cdot \text{OPT} \quad \Box
\]

4 Primal-Dual Algorithm

The primal-dual algorithm is inspired by Gandhi et al. [20]. In the algorithm below a feasible schedule is built iteratively. 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. Let \( \kappa \) be some constant that will be optimized later. If \( r_j > \kappa \cdot p(J') \) we raise the dual variable \( \alpha_j \) until the dual constraint for \( j \) becomes tight. We then schedule \( j \) to be processed before every previously scheduled job.

If \( r_j \leq \kappa \cdot p(J') \) 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 before every previously scheduled job.

**Algorithm 2 Primal-Dual**

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

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 \( r_j \leq \kappa \cdot p(J) \) the job with lowest \( \frac{w_j}{p_j} \) value is removed.

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 $\beta_j$ denote $\beta_{S_j}$ for convenience.

**Lemma 4.** The following are properties of our algorithm.

(a) Every nonzero $\beta_{S_j}$ variable can be written as $\beta_j$ for some job $j$.
(b) For every set $S_j$ that has a nonzero $\beta_j$ variable, if $i \leq j$ then $r_i \leq \kappa \cdot p(S_j)$.
(c) For every job $j$ that has a nonzero $\alpha_j$ variable, $r_j > \kappa \cdot p(S_j)$.
(d) For every job $j$ that has a nonzero $\alpha_j$ variable, if $i \leq j$ then $r_i \leq r_j$.

Each of these observations can easily be verified. We will also need two lemmas that relate our algorithm to the constructed feasible dual solution.

**Lemma 5.** 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.

**Lemma 6.** 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.

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

*Proof.* We use Lemma 5 to rewrite the cost of our solution in terms of the dual variables.

\[
\text{Cost} = \sum_{j \in J} w_j C_j = \sum_{j \in J} (\alpha_j + p_j \sum_{k \geq j} \beta_k) C_j
\]
\[
= \sum_{j \in J} \alpha_j C_j + \sum_{j \in J} p_j \sum_{k \geq j} \beta_k C_j \quad (1)
\]

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

\[
\sum_{j \in J} \alpha_j C_j \leq \sum_{j \in J} \alpha_j (\max_{i \leq j} \{r_i\} + p(S_j)) \quad \text{(Using Lemma 6)}
\]
\[
\leq \sum_{j \in J} \alpha_j (r_j + p(S_j)) \quad \text{(Using (d) of Lemma 4)}
\]
\[
< (1 + \frac{1}{\kappa}) \sum_{j \in J} \alpha_j (r_j + p_j) \quad (2) \text{(Using (c) of Lemma 4)}
\]
Now we bound \( \sum_{j \in J} p_j \sum_{k \geq j} \beta_k C_j \).

\[
\sum_{j \in J} p_j \sum_{k \geq j} \beta_k C_j \leq \sum_{j \in J} p_j \sum_{k \geq j} \beta_k \left( \max_{i \leq j} \{ r_i \} + p(S_j) \right) \quad \text{(Using Lemma 6)}
\]

\[
\leq \sum_{k \in J} \beta_k \sum_{j \leq k} p_j \left( \max_{i \leq j} \{ r_i \} + p(S_j) \right)
\]

\[
\leq \kappa \sum_{k \in J} \beta_k p(S_k) \sum_{j \leq k} p_j + \sum_{k \in J} \beta_k \sum_{j \leq k} p_j p(S_j) \quad \text{(Using (b) of Lemma 4)}
\]

\[
= \kappa \sum_{k \in J} \beta_k p(S_k)^2 + \sum_{k \in J} \beta_k \left( \frac{p(S_k)^2 + p^2(S_k)}{2} \right)
\]

\[
\leq (2\kappa + 1) \sum_{k \in J} \beta_k \left( \frac{p(S_k)^2 + p^2(S_k)}{2} \right) \quad \text{(3)}
\]

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

\[
\text{Cost} \leq (1 + \frac{1}{\kappa}) \sum_{j \in J} \alpha_j (r_j + p_j) + (2\kappa + 1) \sum_{k \in J} \beta_k \left( \frac{p(S_k)^2 + p^2(S_k)}{2} \right)
\]

To get the best approximation guarantee we optimize \( \kappa \). \( \kappa \) 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.

\[
\text{Cost} \leq (1 + \sqrt{2}) \left( \sum_{j \in J} \alpha_j (r_j + p_j) + \sum_{k \in J} \beta_k \left( \frac{p(S_k)^2 + p^2(S_k)}{2} \right) \right)
\]

\[
\leq (1 + \sqrt{2}) \cdot \text{OPT} \quad \Box
\]

References


