5. Weighted number of late jobs

5.1. Release dates and due dates: maximizing the weight of on-time jobs

Once we add release dates, minimizing the number of late jobs becomes a significantly harder problem. For example, we have already seen (in Chapter 3) that the 3-Partition problem can be reduced to the problem of deciding if all jobs can be completed by specified due dates, when subject only to release date constraints. Therefore, deciding if the optimal (weighted) number of late jobs is equal to 0 is strongly NP-hard; this essentially rules out the possibility of approximation algorithms for \(1|r_j|\sum w_j U_j\) (or even \(1|r_j|\sum U_j\)). Instead, we will focus on the equivalent maximization problem: given release and due dates, maximize the total weight of jobs scheduled to complete by their due date. We will give a pseudopolynomial-time algorithm that finds a schedule of objective function value within a factor of 2 of optimal, and then show how it can be adapted to yield, for any \(\varepsilon > 0\), a \((2 + \varepsilon)\)-approximation algorithm for this problem.

The algorithm that we shall present is a primal-dual algorithm. That is, the algorithm simultaneously computes a feasible schedule, as well as a feasible dual solution to a linear programming relaxation of a natural integer programming formulation to the problem; the primal and dual objective function values are within a factor of two of each other, and hence the schedule has total weight at least half the optimal weight. In fact, the algorithm will be presented without explicit reference to any integer programming formulation, its linear relaxation, or its dual. However, after the algorithm has been fully presented and analyzed we will return to link it to a primal-dual approach.

The key mechanism in the algorithm is a technical lemma known as the local ratio principle.

**Lemma 5.1 [Local Ratio Principle].** Consider a linear optimization problem with objective function \(\sum_j w_j x_j\) in decision variables \(x\). Decompose \(w = w^{(1)} + w^{(2)}\), and consider three related optimization problems (over the same class of feasible solutions), with objective functions given by the vectors \(w, w^{(1)},\) and \(w^{(2)}\). If \(\hat{x}\) is a feasible solution of objective function value within a factor \(\rho\) for both \(w^{(1)}\) and \(w^{(2)}\), then it is also within a factor of \(\rho\) for \(w\).

**Proof.** For simplicity of notation, assume that the optimization problem is a maximization problem. Let \(x^*, x^{(1)}, \) and \(x^{(2)}\) be the optimal solutions with respect to \(w, w^{(1)}, \) and \(w^{(2)}\), respectively. We are given that

\[ w^{(1)} \hat{x} \geq w^{(1)} x^{(1)}/\rho \ \text{and} \ w^{(2)} \hat{x} \geq w^{(2)} x^{(2)}/\rho. \]

Hence,

\[ w\hat{x} = (w^{(1)} + w^{(2)}) \hat{x} \geq w^{(1)} x^{(1)}/\rho + w^{(2)} x^{(2)}/\rho \geq w^{(1)} x^*/\rho + w^{(2)} x^*/\rho = (w^{(1)} + w^{(2)}) x^*/\rho = wx^*/\rho, \]

where the second inequality follows from the fact that \(x^*\) is clearly no better than the optimal solutions to the decomposed objective functions \(w^{(1)}\) and \(w^{(2)}\). But this is
5.1. Release dates and due dates: maximizing the weight of on-time jobs

exactly what we claimed: $\mathcal{F}$ has objective function value for $w$ within a factor of $\rho$ for the $w$-optimal solution. $\square$

We shall present a recursive algorithm for the generalization of this problem in which there is a weight $w_j$ associated with each on-time completion time $t$ for job $j$; that is, $t = r_j + p_j, \ldots, d_j$ (where we can assume without loss of generality that $r_j + p_j \leq d_j$, since otherwise we could delete job $j$ altogether). The algorithm will work in time polynomial in $n$ and $D = \max_j d_j$ for any input weights, but we will use the original assumptions about the weights when converting to a polynomial-time algorithm.

We shall view the algorithm as maintaining a set $S$ of pairs $(j, t)$ corresponding to those possible times $t$ at which job $j$ may be scheduled to complete. The algorithm is quite straightforward to state, and it has input consisting of a set $S$ with a corresponding weight vector $w$.

Step 1 Delete all pairs $(j, t)$ from $S$ for which $w_j t \leq 0$. If $S = \emptyset$, then return $\emptyset$.

Step 2 Identify the pair in $S$ that corresponds to the earliest ending point; let $(j^*, t^*)$ denote that pair. (If there is more than one pair with $t^*$, choose among them arbitrarily.) Let $w^* = w_{j^* t'}$. Let $O$ denote the set of pairs $(j, t)$ such that $j \neq j^*$, and scheduling $j$ to finish at $t$ would overlap the processing of $j^*$ to finish at $t^*$ (or equivalently, $[t - p_j, t) \cap [t^* - p_{j^*}, t^*) \neq \emptyset$.) Define

$$w^{(1)}_{jt} = \begin{cases} w^*, & \text{if } j = j^* \text{ or } (j, t) \in O, \\ 0, & \text{otherwise}, \end{cases}$$

and let $w^{(2)} = w - w^{(1)}$.

Step 3 Call the algorithm recursively with the current set $S$ and $w^{(2)}$. Let $\mathcal{F}'$ denote the set returned by the algorithm.

Step 4 If $(j^*, t^*)$ can be added to the solution $\mathcal{F}'$ returned by the algorithm and still be a feasible solution, let $\mathcal{F} \leftarrow \mathcal{F}' \cup \{(j^*, t^*)\}$; otherwise, let $\mathcal{F} \leftarrow \mathcal{F}'$. The algorithm returns $\mathcal{F}$.

It is easy to see inductively that the pairs output by the algorithm is a feasible selection. We first show that any feasible solution is a 2-approximation for the specially constructed objective function $w^{(1)}$.

Lemma 5.2. Consider any feasible selection of intervals $\mathcal{F}$. Then, (i) there exists at most one pair $(j, t) \in \mathcal{F}$ such that $j = j^*$, and (ii) there exists at most one pair $(j, t) \in \mathcal{F}$ such that $(j, t) \in O$.

Proof. The first claim is trivial, since each job is scheduled at most once in any feasible schedule. For the second part of the claim, observe that since we have selected the interval $(j^*, t^*)$ such that $t^* \leq t$ for any other pair in $S$, it follows that if $(j, t) \in O$ then the corresponding interval $[t - p_j, t)$ intersects $[t^* - p_{j^*}, t^*)$ at the time
5. Weighted number of late jobs

immediately preceding \( t^* \). But since \( \mathcal{F} \) is feasible, there can be only one such pair \((j, t)\), since then the job \( j \) is being processed immediately prior to \( t^* \). □

**Corollary 5.3.** For the input \( S \) considered in Step 2, the feasible solution \( \mathcal{F} \) constructed by the algorithm is of \( w^{(1)} \) objective function value within a factor of 2 of the optimal value for \( w^{(1)} \).

**Proof.** Either the algorithm schedules \((j^*, t^*)\) or else set \( \mathcal{F}' \) returned in Step 3 contains a pair \((j, t)\) with \( j = j^* \) or in \( j \in O \). In either case, the feasible solution \( \mathcal{F} \) computed by the algorithm has \( w^{(1)} \) weight at least \( w' \) (and in fact, it is exactly \( w' \)). However, Lemma 5.2 directly implies that any feasible solution has weight at most \( 2w' \), and hence the claim follows. □

**Theorem 5.4.** The recursive local ratio-based algorithm finds a feasible solution of total weight that is at least half the optimal weight for the given input weights \( w \) with the given feasible selection set \( S \).

**Proof.** The proof is by induction on the number of recursive calls of the algorithm. If there are no recursive calls, then there are no pairs in \( S \) of positive weight; hence the optimal value is 0, and the algorithm outputs the empty set, a selection of objective function value 0. We suppose now that the algorithm finds a solution of objective function value at least half the optimal whenever the number of recursive calls is at most \( \ell \), and consider an execution of the algorithm that requires \( \ell + 1 \) recursive calls. This execution computes a set \( \mathcal{F}' \) and a set \( \mathcal{F} \). By induction, \( \mathcal{F}' \) has weight at least half the optimal value for the input at Step 2 for weights \( w^{(2)} \). Since \( w^{(2)}_{j^* t^*} = 0 \), the weight with respect to \( w^{(2)} \) of \( \mathcal{F} \) must be the same as \( \mathcal{F}' \), and hence is at least half the optimal value for the input at Step 2. By Corollary 5.3, the solution \( \mathcal{F} \) is at least half the optimal value with respect to the weights \( w^{(1)} \). But then, by the local ratio principle, it is at least half the optimal value with respect to the weights \( w \). And clearly, deleting possible selections of non-positive weight does not change the optimal value, and so the theorem follows. □

We still need to consider the running time of the algorithm. To prove a pseudopolynomial time bound is relatively straightforward. If \( D = \max_j d_j \), the set \( S \) contains at most \( nD \) pairs initially. With each recursive call, at least one additional pair, that is, \((j^*, t^*)\), has weight 0, and hence is deleted in Step 1. So there are at most \( nD \) recursive calls, and each can be implemented in \( O(nD) \) time (updating the weight for each active pair).

There is a simple trick that converts this algorithm to one that runs in polynomial time. Suppose that we change Step 1 to delete a pair \((j, t)\) when its weight is less than \( \epsilon w_j \). Now, each time job \( j = j^* \), all of the pairs remaining for job \( j \) have weight that decreases by \( \epsilon w_j \). Hence, job \( j \) is selected as \( j^* \) at most \( 1/\epsilon \) times throughout all recursive calls of the algorithm, and so there are at most \( n/\epsilon \) recursive calls overall. However, we must also bound the work needed to maintain the weight of each active pair; if we keep this data explicitly, we will require work proportional to \( nD \), and this...
5.1. Release dates and due dates: maximizing the weight of on-time jobs

is not polynomial. We will argue that the weights are quite simple to maintain, and that each recursive call increases the number of distinct weights for job $j$ by at most 1.

We will focus on the pairs active in Step 2, and show that if we view $w_{jt}$ for each job $j$ as a function of $t$, this function is a positive monotone non-decreasing step function over some interval $[r, d_j]$; we will show this by induction on the number of recursive calls, and further show that the number of “steps” in this function increases by at most 1 with each recursive call. Suppose that after some number of recursive calls, the weights for $j$ are a step function attaining values

$$W_1 < W_2 < \cdots < W_k$$

with endpoints of these intervals

$$r = T_1 < T_2 < \cdots < T_k < T_{k+1} = d_j;$$

that is, the weight for $(j, t)$ is $W_i$ for $t = T_i + 1, \ldots, T_{i+1}$. (There is the degenerate case that the weight for $t = r$ is also $W_1$.) Clearly, this is true initially, where the weight of $(j, t)$ is $w_j$ for $t = r_j + p_j, \ldots, d_j$. For any pair $(j, t)$ with positive weight in Step 2, we know that $t^* \leq t$ (by the choice of $(j^*, t^*)$). If $j = j^*$, the effect of Step 2 is to decrease all of the values by $w^*$. Thus, the first step “disappears” and in fact there is at least one fewer step in the resulting step function for this job; more steps might also disappear due to the modification that pairs with weight less than $\varepsilon w_j$ are deleted in Step 1. If $j \neq j^*$, then for $t$ such that $t^* > t - p_j$ we subtract $w^*$ from that weight $w_{jt}$. But this means that for an initial part of the step function we decrease the values of the steps. The worst-case setting is that each of the weights remains at least $\varepsilon w_j$, and we cause one step to be subdivided into two.

Combining the previous two structural results, we know that the step function for each job always has at most $n/\varepsilon$ steps, and so we can implement each iteration to run in $O(n^2/\varepsilon)$ time, and so the entire algorithm can be implemented to run in $O(n^3/\varepsilon^2)$ time.

But does the modified algorithm still produce a good solution? We use the modified algorithm to construct a new objective function: let $\hat{w}$ denote the weight function such that $w = \hat{w}$ except when the pair $(j, t)$ is deleted due to its weight becoming less than $\varepsilon w_j$; in this case, set $\hat{w}_{jt} = w_{jt} - \varepsilon w_j$. The crucial observation is that if we run the “correct” algorithm on $\hat{w}$, and the modified algorithm on $w$, then the two executions are identical, and we must obtain the same result. Thus, the solution $\hat{F}$ that is output by both algorithms has weight at least half of the optimal for $\hat{w}$. Let $W^*$ denote the optimal value for the original input $w$. If we evaluate an optimal solution for $w$ with respect to $\hat{w}$, then we get at least $(1 - \varepsilon)W^*$. Hence the optimal solution for $\hat{w}$ is at least $(1 - \varepsilon)W^*$, and the solution found by the algorithm has weight (with respect to $\hat{w}$) at least $(1 - \varepsilon)W^*/2$. But then it must also have value with respect to $w$ at least this much, and we have proved the following theorem.

**Theorem 5.5.** For any $\varepsilon > 0$, the local ratio principle yields a $(2 + \varepsilon)$-approximation
algorithm for $1|r_j|\sum w_j \bar{U}_j$.

Finally, we shall explain the connection between this algorithm and the notion of a primal-dual algorithm. Consider the integer programming formulation for this problem in which there is a variable $x_{jt}$ for each pair $(j,t) \in S$; this is a 0-1 variable in which the value 1 indicates that job $j$ is scheduled to complete at time $t$. We can give the following formulation:

$$\text{maximize } \sum_{j=1}^{n} \sum_{t=1}^{D} w_j x_{jt}$$
$$\text{subject to } \sum_{t=1}^{D} x_{jt} \leq 1, \text{ for each } j = 1, \ldots, n,$$
$$\sum_{j=1}^{n} \sum_{s=t}^{\min(D,t+p_j-1)} x_{js} \leq 1, \text{ for each } t = 1, \ldots, D,$$
$$x_{jt} = 0, \text{ for each } j = 1, \ldots, n, t \notin \{r_j + p_j, \ldots, d_j\},$$
$$x_{jt} \in \{0,1\}$$

The linear programming relaxation of this formulation has the following dual linear program, where $u_j$ is the dual variable corresponding to job $j$ for the first set of constraints, and $v_t$ is the dual variable corresponding to time $t$ in the second set of constraints:

$$\text{minimize } \sum_{j=1}^{n} u_j + \sum_{t=1}^{D} v_t$$
$$\text{subject to } u_j + \sum_{s=t-p_j+1}^{t} v_s \geq w_j, \text{ for each } j = 1, \ldots, n, t = 1, \ldots, D,$$
$$u_j \geq 0, \text{ for each } j = 1, \ldots, n,$$
$$v_t \geq 0, \text{ for each } t = 1, \ldots, D.$$

A slight modification of the algorithm can be viewed as constructing a feasible dual solution in the following way: when $(j^*,t^*)$ is identified, we increase both $u_{j^*}$ and $v_{t^*}$ by $w^*/2$. We modify the algorithm to keep track of the dual constraints $(j,t)$ that are already satisfied by the dual solution (rather than those pairs whose modified weight is non-negative). The modified algorithm is therefore guaranteed to produce a feasible dual solution. Furthermore, it is straightforward to see that in each level of recursion, the dual feasible solution increases in objective function value, and that $w^*/2$ of the increase can be viewed as partially paying for a pair that is selected for the final solution; this gives an alternate way to state the essence of the 2-approximation result.