Scheduling Unit Jobs with a Common Deadline to Minimize the Sum of Weighted Completion Times and Rejection Penalties *

Nevzat Onur Domaniç † C. Gregory Plaxton †

September 2014

Abstract

We study the problem of scheduling unit jobs on a single machine with a common deadline where some jobs may be rejected. Each job has a weight and a profit and the objective is to minimize the sum of the weighted completion times of the scheduled jobs plus the sum of the profits of the rejected jobs. Our main result is an $O(n \log n)$-time algorithm for this problem. In addition, we show how to incorporate weighted tardiness penalties with respect to a common due date into the objective while preserving the $O(n \log n)$ time bound. We also discuss connections to a special class of unit-demand auctions. Finally, we establish that certain natural variations of the scheduling problems that we study are NP-hard.

*This research was supported by NSF Grant CCF–1217980.

†Department of Computer Science, University of Texas at Austin, 2317 Speedway, Stop D9500, Austin, Texas 78712–1757. Email: {onur,plaxton}@cs.utexas.edu

1 Introduction

In many scheduling problems, we are given a set of jobs, and our goal is to design a schedule for executing the entire set of jobs that optimizes a particular scheduling criterion. Scheduling with rejection, however, allows some jobs to be rejected, either to meet deadlines or to optimize the scheduling criterion, while possibly incurring penalties for rejected jobs. In this paper we study the problem of scheduling unit jobs (i.e., jobs with an execution requirement of one time unit) with individual weights (w_i) and profits (e_i) on a single machine with a common deadline ($\overline{d}$) where some jobs may be rejected. If a job is scheduled by the deadline then its completion time is denoted by C_i; otherwise it is considered rejected. Let S denote the set of scheduled jobs and $\overline{S}$ denote the set of rejected jobs. The goal is to minimize the sum of the weighted completion times of the scheduled jobs plus the total profits of the rejected jobs. Hence job profits can be equivalently interpreted as rejection penalties. We represent the problem using the scheduling notation introduced by Graham et al. [10] as:

$$1 \mid p_i = 1, \overline{d}_i = \overline{d} \mid \sum_S w_i C_i + \sum_{\overline{S}} e_i \quad . \tag{1}$$

We assume that the number of jobs is at least $\overline{d}$. If not, letting U_+ (resp., U_-) denoting the set of the jobs with nonnegative (resp., negative) weights, it is easy to observe that there exists a solution in which each job in U_+ (resp., U_-) that is not rejected is scheduled in one of the first $|U_+|$ (resp., last $|U_-|$) slots. Using this observation, we can solve the given instance by solving two smaller instances for which our assumption is satisfied.

Like many other scheduling problems involving unit jobs, Problem 1 can be solved in polynomial time by a reduction to the maximum weight matching problem in bipartite graphs. Our contribution is an $O(n \log n)$-time algorithm for Problem 1 where n denotes the number of jobs. Engels et al. [6] give a pseudo-polynomial-time dynamic programming algorithm for the same objective except that variable processing times are allowed and no deadline restriction is imposed. Engels et al. first show that the decision version of the problem is NP-complete and then give an FPTAS. They also remark that a running time of $O(n^2)$ can be achieved for the special case of unit processing times; our work improves this bound to $O(n \log n)$.

More general cases of Problem 1, almost all dealing with variable processing times, have been studied extensively. One of the earliest works that considers job specific profits and lateness penalties [23] reduces to our problem in the special case of setting all processing times to 1 and all due dates to 0. Two recent surveys review the research on various scheduling problems in which it is typically necessary to reject some of the jobs in order to achieve optimality [19, 22]. Epstein et al. [7] focus on unit jobs but consider only the online version of the problem. Shabtay et al. [18] split the scheduling objective into two criteria: the scheduling cost, which depends on the completion times of the jobs, and the rejection cost, which is the sum of the penalties paid for the rejected jobs. In addition to optimizing the sum of these two criteria, the authors study other variations of the problem such as optimizing one criterion while constraining the other, or identifying all Pareto-optimal solutions for the two criteria. The scheduling cost in that work is not exactly the weighted sum of the completion times; however, several other similar objectives are considered. We show that our problem becomes NP-hard if we split our criteria in the same manner and aim for optimizing one while bounding the other.

Given the improvement in running time that we achieve for Problem 1, it is natural to ask whether our approach can be adapted to obtain fast algorithms for interesting variants of Problem 1. We show the following generalization of Problem 1 can also be solved in $O(n \log n)$ time:

$$1 \mid p_i = 1, d_i = d, \overline{d}_i = \overline{d} \mid \sum_{S} w_i C_i + c \sum_{S} w_i T_i + \sum_{\overline{S}} e_i \ . \tag{2}$$

In Problem 2, every job also has a common due date d, and completing a job after the due date incurs an additional tardiness penalty that depends on its weight and a positive constant c. The tardiness of a job is defined as $T_i = \max\{0, C_i - d\}$. Similar to Problem 1, we assume that the number of jobs is at least $\overline{d}$.

We solve Problems 1 and 2 by finding a maximum weight matching (MWM) in a complete bipartite graph that represents the scheduling instance. Due to the special structure of the edge weights, the space required to represent this graph is linear in the number of vertices. Thus, aside from the scheduling applications, this work contributes to the research aimed at developing quasi-linear algorithms for matching problems in compactly representable bipartite graphs. Both unweighted and vertex-weighted matching problems in convex bipartite graphs, the graphs in which the right vertices can be enumerated such that the neighbors of each left vertex are consecutive, have been studied extensively [8, 9, 12, 14, 24]. Plaxton [15] studies vertex-weighted matchings in two-directional orthogonal ray graphs, which generalize convex bipartite graphs. In contrast, the current paper focuses on a class of compactly representable bipartite graphs that is simpler in terms of the underlying graph structure (all edges are present), but allows for more complex edge weights.

The cost (distance) matrix of the complete bipartite graph that we construct for solving Problems 1 and 2 is a Monge matrix. An $n \times m$ matrix $C = (c_{ij})$ is called a Monge matrix if $c_{ij} + c_{rs} \leq c_{is} + c_{rj}$ for $1 \leq i < r \leq n$, $1 \leq j < s \leq m$. Burkard [2] provides a survey of the rich literature on applications of Monge structures in combinatorial optimization problems. When the cost matrix of a bipartite graph is a Monge matrix, an optimal maximum cardinality matching can be found in $O(nm)$ time where n is the number of rows and m is the number of columns. If $n = m$ then the diagonal of the cost matrix is a trivial solution. Aggarwal et al. [1] study several weighted bipartite matching problems where, aside being a Monge matrix, additional structural properties are assumed for the cost matrix. The authors present an $O(n \log m)$-time divide and conquer algorithm for the case where the number of rows n is at most the number of columns m and each row is bitonic, i.e., each row is a non-increasing sequence followed by a non-decreasing sequence. If we represent the edge weights of the bipartite graph that we construct for solving our problems in a matrix so that the rows correspond to the jobs and the columns correspond to the time slots, then both the Monge property and the bitonicity property are satisfied; in fact each row is monotonic. However, we end up having more rows than columns, which renders the algorithm of [1] inapplicable for our problems. If we had more columns than rows, as assumed in [1], then we would have a trivial solution which could be constructed by sorting the jobs with respect to their weights. In summary, similar to [1], our algorithm efficiently solves the weighted bipartite matching problem for Monge matrices having an additional structure on the rows. In contrast, the structural assumption we place on the rows is stronger than that of [1], and we require more rows than columns, whereas [1] requires the opposite.

Another application of bipartite graphs is in the context of unit-demand auctions. In a unit-demand auction, a collection of items is to be distributed among several bidders and each bidder is to receive at most one item [5, 13, 20]. Each bidder has a private value for each item, and submits to the auction a unit-demand bid that specifies a separate offer for each item. The VCG mechanism can be used to determine the outcome of a unit-demand auction, i.e., allocation and pricing of the items. The VCG allocation corresponds to an (arbitrary) MWM of the bipartite graph in which each left vertex represents a bid, each right vertex represents an item, and the weight of the edge from a bid u to an item v represents the offer of the bid u for item v. The VCG mechanism is known to enjoy a number of desirable properties including efficiency, envy-freedom, and strategyproofness. Another contribution of this paper is an $O(n \log n)$-time algorithm for computing the VCG prices, given a VCG allocation of an auction instance that can be represented by a more general class of the complete bipartite graphs than the ones that we construct to solve Problems 1 and 2.

Organization. Section 2 describes the fast $O(n \log n)$-time algorithm for Problem 1. Some of the proofs and a brief implementation are deferred to App. A. Section 3 and App. B describe how to extend the algorithm to solve Problem 2 within the same time bound. Section 4 and App. C view the problem from a unit-demand auction perspective and present an $O(n \log n)$-time algorithm for computing the VCG prices. Finally, App. E proves the NP-hardness of the bicriteria variations of Problem 1 via reductions from the partition problem.

2 A Fast Algorithm for Problem 1

We encode an instance of Problem 1 as a weighted matching problem on a graph drawn from a certain family. Below we define this family, which we call $\mathcal{G}$, and we discuss how to express an instance of Problem 1 in terms of a graph in $\mathcal{G}$.

We define $\mathcal{G}$ as the family of all complete edge-weighted bipartite graphs $G = (U, V, w)$ such that the following conditions hold: $|U| \geq |V|$; each left vertex u in U has two associated integers $u.\mathit{profit}$ and $u.\mathit{priority}$; the left vertices are indexed from 1 in non-decreasing order of priorities, breaking ties arbitrarily; right vertices are indexed from 1; the weight $w(u, v)$ of the edge between a left vertex u and a right vertex v is equal to $u.\mathit{profit} + u.\mathit{priority} \cdot j$ where j denotes the index of v. Note that a graph $G = (U, V, w)$ in $\mathcal{G}$ admits an $O(|U|)$-space representation.

Let I be an instance of Problem 1. The instance I consists of a set of n jobs to schedule, each with a profit and a weight, and a common deadline $\overline{d}$ where we assume that $n \geq \overline{d}$ as discussed in Sect. 1. We encode the instance I as a graph $G = (U, V, w)$ in $\mathcal{G}$ such that the following conditions hold: $|U| = n$; $|V| = \overline{d}$; each left vertex represents a distinct job in I; each right vertex represents a time slot in which a job in I can be scheduled; for each job in I and the vertex u that represents that job, $u.\mathit{profit}$ is equal to the profit of the job and $u.\mathit{priority}$ is equal to the negated weight of the job. It is easy to see by inspecting the objective of Problem 1 that minimizing the weighted sum of completion times is equivalent to maximizing the same expression with negated weights, and minimizing the sum of the profits of the rejected jobs is equivalent to maximizing the sum of the profits of the scheduled jobs. Hence, instance I of Problem 1 is equivalent to the problem of finding a maximum weight matching (MWM) of a graph $G = (U, V, w)$ in $\mathcal{G}$ that encodes I. Given this correspondence between the two problems, we refer to the left vertices (resp., right vertices) of a graph in $\mathcal{G}$ as *jobs* (resp., *slots*). The problem of computing an MWM of a graph $G = (U, V, w)$ in $\mathcal{G}$ can be reduced to the maximum weight maximum cardinality matching (MWMCM) problem by adding $|V|$ dummy jobs, each with profit and priority zero, to obtain a graph that also belongs to $\mathcal{G}$.

As a result of the equivalence of the two problems mentioned above and the reduction from the MWM to the MWMCM problem, we can obtain an $O(n \log n)$-time algorithm for Problem 1 by providing an $O(|U| \log |U|)$-time algorithm to compute an MWMCM of a graph $G = (U, V, w)$ in $\mathcal{G}$. Before discussing this algorithm further, we introduce some useful definitions.

Let $G = (U, V, w)$ be a graph in $\mathcal{G}$. We say that a subset U' of U is *optimal* for G if there exists an MWMCM M of G such that the set of jobs that are matched in M is equal to U'. Lemma 1 below shows that it is straightforward to efficiently construct an MWMCM of G given an optimal set of jobs for G. Let U' be a subset of U with size $|V|$ and let $i_1 < \cdots < i_{|V|}$ denote the indices of the jobs in U'. Then we define $\mathit{matching}(U')$ as the set of $|V|$ job-slot pairs obtained by pairing the job with index i_k to the slot with index k for $1 \leq k \leq |V|$. The following lemma is a straightforward application of the rearrangement inequality [11, Section 10.2, Theorem 368] to our setting.

Lemma 1. Let $G = (U, V, w)$ be a graph in $\mathcal{G}$. Let U' be a subset of U with size $|V|$. Let W denote the maximum weight of any MCM of G that matches U'. Then $\mathit{matching}(U')$ is of weight W.

Having established Lemma 1, it remains to show how to efficiently identify an optimal set of jobs for a given graph $G = (U, V, w)$ in $\mathcal{G}$. The main technical result of this section is an $O(|U| \log |U|)$-time dynamic programming algorithm for accomplishing this task. The following definitions are useful for describing our dynamic programming framework.

Let $G = (U, V, w)$ be a graph in $\mathcal{G}$. For any integer i such that $0 \leq i \leq |U|$, we define U_i as the set of jobs with indices 1 through i. Similarly, for any integer j such that $0 \leq j \leq |V|$, we define V_j as the set of slots with indices 1 through j. For any integers i and j such that $0 \leq j \leq i \leq |U|$ and $j \leq |V|$, we define $G_{i,j}$ as the subgraph of G induced by the vertices $U_i \cup V_j$, and we define $W(i, j)$ as the weight of an MWMCM of $G_{i,j}$. Note that any subgraph $G_{i,j}$ of G also belongs to $\mathcal{G}$.

Let us define $\mathcal{G}^*$ as the family of all graphs in $\mathcal{G}$ having an equal number of slots and jobs. Given a graph $G = (U, V, w)$ in $\mathcal{G}^*$, our dynamic programming algorithm computes in $O(|U| \log |U|)$ total time an optimal set of jobs for each $G_{|U|,j}$ for $1 \leq j \leq |U|$. For any graph $G' = (U, V', w')$ in $\mathcal{G}$, we can construct a graph $G = (U, V, w)$ in $\mathcal{G}^*$ satisfying $G'_{|U|,j} = G_{|U|,j}$ for all $1 \leq j \leq |V'|$ by defining V as the set of $|U|$ slots indexed from 1 through $|U|$. Thus, given any graph $G' = (U, V', w')$ in $\mathcal{G}$, our algorithm can be used to identify an optimal set of jobs for each subgraph $G'_{|U|,j}$ for $1 \leq j \leq |V'|$ in $O(|U| \log |U|)$ total time.

Throughout the remainder of this section, we fix a graph instance $G = (U, V, w)$ in $\mathcal{G}^*$. The presentation of the algorithm is organized as follows. Section 2.1 introduces the core concept, which we call the *acceptance order*, that our algorithm is built on. Section 2.2 presents the key idea (Lemma 5) underlying our algorithm for computing the acceptance order. Finally, Sect. 2.3 describes an efficient augmented binary search tree implementation of the algorithm.

2.1 Acceptance Orders

Lemma 1 reduces Problem 1 to the problem of identifying an optimal subset of U for G. In addition to an optimal set of jobs for G, our algorithm determines for each integer i and j such that $0 \leq j \leq i \leq |U|$, a subset $\mathit{best}(i, j)$ of U_i that is optimal for $G_{i,j}$ (Lemma 3). There are quadratically many such sets, so in order to run in quasilinear time, we compute a compact representation of those sets by exploiting the following two properties. The first property is that

$best(i, j-1)$ is a subset of $best(i, j)$ for $1 \leq j \leq i \leq |U|$. Thus, for a fixed i, the sequence of sets $best(i, 1), \ldots, best(i, i)$ induces an ordering σ_i of jobs U_i, which we later define as the acceptance order of U_i, where the job at position j of σ_i is the one that is present in $best(i, j)$ but not in $best(i, j-1)$. The second property is that σ_{i-1} is a subsequence of σ_i for $1 \leq i \leq |U|$. This second property suggests an incremental computation of σ_i's which will be exploited to find the weights of MWMCMs for all prefixes of jobs to solve Problem 2, as described in Sect. 3.

We now give the formal definitions of the acceptance order and the optimal set $best(i, j)$, and present two associated lemmas. The proofs of these two lemmas are provided in App. A.1.

We say that a vertex is *essential* for an edge-weighted bipartite graph G if it belongs to every MWMCM of G.

For any integer i such that $0 \leq i \leq |U|$ we define σ_i inductively as follows: σ_0 is the empty sequence; for $i > 0$ let u denote the job with index i, then σ_i is obtained from σ_{i-1} by inserting job u immediately after the prefix of σ_{i-1} of length $p - 1$ where p, which we call the position of u in σ_i, is the minimum positive integer such that job u is essential for $G_{i,p}$. It is easy to see that σ_i is a sequence of length i and that $1 \leq p \leq i$ since u is trivially essential for $G_{i,i}$. Furthermore, σ_{i-1} is a subsequence of σ_i for $1 \leq i \leq |U|$, as claimed above.

We say that σ_i is the *acceptance order* of the set of jobs U_i. Note that $\sigma_{|U|}$ is the acceptance order of the set of all jobs.

Lemma 2. Let i and j be any integers such that $1 \leq j \leq i \leq |U|$ and let u denote the job with index i. Then job u is essential for $G_{i,j}$ if and only if the position of u in σ_i is at most j.

For any integers i and j such that $0 \leq j \leq i \leq |U|$, we define $best(i, j)$ as the set of the first j jobs in σ_i. Thus, $best(i, j-1)$ is a subset of $best(i, j)$ for $1 \leq j \leq i \leq |U|$, as claimed above.

Lemma 3. Let i and j be any integers such that $0 \leq j \leq i \leq |U|$. Then $matching(best(i, j))$ is an MWMCM of $G_{i,j}$.

Lemmas 1 and 3 imply that once we compute the acceptance order $\sigma_{|U|}$, we can sort its first $\overline{d}$ jobs by their indices to obtain a matching to solve Problem 1.

2.2 Computing the Acceptance Order

As we have established the importance of the acceptance order $\sigma_{|U|}$, we now describe how to compute it efficiently. We start with σ_1 and introduce the tasks one by one in index order to compute the sequences $\sigma_2, \ldots, \sigma_{|U|}$ incrementally. Once we know σ_{i-1}, we just need to find out where to insert the job with index i in order to compute σ_i. We first introduce some definitions and a lemma, whose proof is provided in App. A.1, and then we describe the key idea (Lemma 5) for finding the position of a job in the corresponding acceptance order.

For any integers i and j such that $1 \leq j \leq i \leq |U|$, let $\sigma_i[j]$ denote the job with position j in σ_i, where $\sigma_i[1]$ is the first job in σ_i.

For any job u that belongs to U, we define $better(u)$ as the set of jobs that precede u in σ_i where i denotes the index of u. Thus $|better(u)| = p - 1$ where p is the position of u in σ_i. The set $better(u)$ is the set of jobs that precede u both in index order and in acceptance order.

Lemma 4. Let i and j be integers such that $1 \leq j \leq i \leq |U|$, and let i' denote the index of job $\sigma_i[j]$. Then the set of jobs in $best(i, j-1)$ with indices less than i' is equal to $better(\sigma_i[j])$.

For any subset U' of U, we define $\mathit{sum}(U')$ as $\sum_{u \in U'} u.\mathit{priority}$.

Now we are ready to discuss the idea behind the efficient computation of the acceptance orders incrementally. Assume that we already know the acceptance order σ_{i-1} of the set of the first $i-1$ jobs for some integer i such that $1 < i \leq |U|$. Let u denote the job with index i. If we can determine in constant time, for any job in the set U_{i-1}, whether u precedes that job in σ_i, then we can perform a binary search in order to find in logarithmic time the position of u in σ_i. Suppose that we would like to know whether u precedes $\sigma_{i-1}[j]$ in σ_i for some integer j such that $1 \leq j < i$. In other words we would like to determine whether the position of u in σ_i is at most j. In what follows, let u' denote the job $\sigma_{i-1}[j]$ and let v denote the slot with index j. Then by Lemma 2, job u precedes u' in σ_i if and only if u is essential for $G_{i,j}$.

In order to determine whether job u is essential for $G_{i,j}$, we need to compare the weight of a heaviest possible matching for $G_{i,j}$ that does not include u to the weight of a heaviest possible matching for $G_{i,j}$ that includes u. The former weight is $W(i-1,j)$. Since job u has the highest index among the jobs with indices 1 through i, by Lemma 1, the latter weight is equal to $w(u,v) + W(i-1,j-1)$.

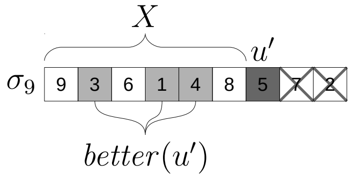
Let X denote $\mathit{best}(i-1,j-1)$. Since $\mathit{best}(i-1,j-1) + u' = \mathit{best}(i-1,j)$, Lemma 3 implies that the weight of $\mathit{matching}(X+u')$ is equal to $W(i-1,j)$. By Lemma 3, the weight of $\mathit{matching}(X)$ is $W(i-1,j-1)$. Since job u has the highest index among the jobs in $X+u$, the weight of $\mathit{matching}(X+u)$ is $w(u,v) + W(i-1,j-1)$.

Combining the results of the preceding paragraphs, we conclude that job u is essential for $G_{i,j}$ if and only if the weight of $\mathit{matching}(X+u)$ is greater than the weight of $\mathit{matching}(X+u')$.

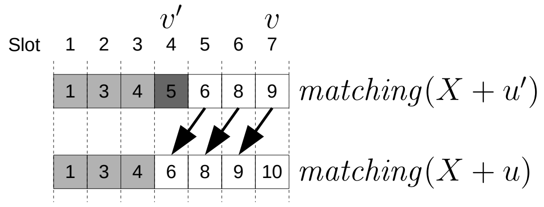
Figure 1 shows an example where $i = 10$ and $j = 7$. Thus we are trying to determine whether the job with index 10 precedes $\sigma_9[7]$ in σ_{10}. In this example, u denotes the job with index 10 and u' denotes $\sigma_9[7]$, which is the job with index 5, as shown in Fig. 1a. The set X is the first 6 jobs in σ_9. The jobs appearing past u' in σ_9, jobs with indices 7 and 2, do not participate in the matchings that we are interested in so they are crossed out. Figure 1b shows the two matchings $\mathit{matching}(X+u')$ and $\mathit{matching}(X+u)$ of which we would like to compare the weights. As seen in Fig. 1b, each job in X with index less than that of job u', shaded light gray in the figure, is matched to the same slot in both $\mathit{matching}(X+u)$ and $\mathit{matching}(X+u')$. By Lemma 4, those jobs are the ones in the set $\mathit{better}(u')$, which are the jobs with indices 1, 3 and 4 in the example. Hence job u' occurs in position $|\mathit{better}(u')|+1$ when we sort the set of jobs $X+u'$ by index and thus it is matched to the slot with index $|\mathit{better}(u')|+1$ in $\mathit{matching}(X+u')$. Moreover, each job in X with index greater than that of job u' is matched to a slot with index one lower in $\mathit{matching}(X+u)$ than in $\mathit{matching}(X+u')$, as depicted by the arrows in Fig. 1b for the jobs with indices 6, 8, and 9.

Hence the weight of $\mathit{matching}(X+u)$ minus the weight of $\mathit{matching}(X+u')$ is equal to $w(u,v) - w(u',v')$ plus the sum of the priorities of all jobs in $\mathit{best}(i-1,j-1)$ with indices greater than that of u', where v' denotes the slot with index $|\mathit{better}(u')|+1$. By Lemma 4, the latter sum is equal to $\mathit{sum}(\mathit{best}(i-1,j-1)) - \mathit{sum}(\mathit{better}(u'))$. These observations establish the proof of the following lemma which we utilize in computing the acceptance orders incrementally.

Lemma 5. Let i and j be integers such that $1 \leq j < i \leq |U|$. Let u denote the job with index i and let u' denote the job $\sigma_{i-1}[j]$. Then the following are equivalent: (1) The position of u in σ_i is at most j; (2) Job u is essential for $G_{i,j}$; (3) The weight of $\mathit{matching}(\mathit{best}(i-1,j-1)+u)$ is greater than the weight of $\mathit{matching}(\mathit{best}(i-1,j-1)+u')$; and (4) $w(u,v) > w(u',v') + \mathit{sum}(\mathit{best}(i-1,j-1)) - \mathit{sum}(\mathit{better}(u'))$ where v denotes the slot with index j and v' denotes the slot with



(a) Acceptance order σ_9



(b) The two matchings to compare

Figure 1: An example in which we try to determine whether the job with index 10 precedes $\sigma_9[7]$ in σ_{10}. Each box represents the job whose index is shown inside.

index $|better(u')| + 1$.

2.3 Binary Search Tree Implementation

We obtain an efficient algorithm utilizing a self-balancing augmented binary search tree (BST) for incrementally computing the acceptance orders by a suitable choice of ordering the jobs, and an augmentation that is crucial in applying Lemma 5 in constant time. The jobs are stored in the BST so that an inorder traversal of the BST yields the acceptance order. The algorithm runs $|U|$ iterations where the job with index i is inserted into the BST at iteration i to obtain σ_i from σ_{i-1} by performing a binary search. We first give some definitions that are useful in the description of the algorithm and then we state in Lemma 6 how to perform the comparisons for the binary search.

For a binary tree T and an integer i such that $1 \leq i \leq |U|$, we define the predicate $ordered(T, i)$ to hold if T contains i nodes that represent the jobs U_i, and the sequence of the associated jobs resulting from an inorder traversal of T is σ_i. The job represented by a node x is denoted by $x.job$.

Let T be a binary tree satisfying $ordered(T, i)$ for some i. For any node x in T, $precede(x, T)$ is defined as the set of jobs associated with the nodes that precede x in an inorder traversal of T.

Lemma 6. Let i be an integer such that $1 < i \leq |U|$ and let u denote the job with index i. Let T be a binary tree satisfying $ordered(T, i-1)$ and let x be a node in T. Assume that $|precede(x, T)|$, $sum(precede(x, T))$, $|better(x.job)|$, and $sum(better(x.job))$ are given. Then we can determine in constant time whether u precedes $x.job$ in σ_i.

Proof. Let j denote $|precede(x, T)| + 1$. Then $ordered(T, i-1)$ implies that $x.job$ is $\sigma_{i-1}[j]$ and $sum(precede(x, T))$ is equal to $sum(best(i-1, j-1))$. Now let u' denote $\sigma_{i-1}[j]$. Then we can test Inequality 4 of Lemma 5 in constant time to determine whether the position of u in σ_i is at most j, thus whether u precedes u' in σ_i. $\square$

Lemma 6 implies that once we know certain quantities about a node x in the BST then we can tell in constant time whether the new job precedes $x.job$ in the acceptance order. The necessary information to compute the first two of those quantities can be maintained by standard BST augmentation techniques as described in [3, Chapter 14]. The other two quantities turn out to be equal to the first two at the time the node is inserted into the BST and they can be stored along with the node. The details are in the proof of the following result, which is presented together with a concise implementation in Apps. A.2 and A.3.

Theorem 1. The acceptance order of U can be computed in $O(|U| \log |U|)$ time.

As mentioned earlier, once $\sigma_{|U|}$ is computed, we can extract an MWMCM of $G_{|U|,j}$ for any j such that $1 \leq j \leq |U|$. If we are only interested in solutions for j up to some given m, then the algorithm can be implemented in $O(n \log m)$ time by keeping at most m nodes in the BST. We achieve this by deleting the rightmost node when the number of nodes exceeds m. Note that if the jobs are not already sorted by priorities then we still need to spend $O(n \log n)$ time.

If we would like to find out the weights of the MWMCMs of $G_{|U|,j}$ for all j such that $1 \leq j \leq |U|$, a naive approach would be to sort all prefixes of $\sigma_{|U|}$ and to compute the weights. Appendix D explains how to compute all those weights incrementally in linear time.

3 Introducing Tardiness Penalties

Given the improvement in running time that we achieve for Problem 1, we consider solving several variations of that problem and other related problems in more general families of compact bipartite graphs than the one we introduced in Sect. 2. A possible variation of Problem 1 is to allow a constant number of jobs to be scheduled in each time slot instead of only one. However, our approach of comparing the weights of two matchings that we illustrate in Fig. 1b fails because only some of the jobs, instead of all, in the set X having indices greater than the job we compare with are shifted to a lower slot. Solving this variation would enable us to address scheduling problems having symmetric earliness and tardiness penalties with respect to a common due date.

Another related problem is finding an MWM in a more general complete bipartite graph family that is still representable in space linear in the number of vertices. Consider the following extension to the complete bipartite graph $G = (U, V, w)$ that is introduced in Sect. 2. For each slot (right vertex) v in V, we introduce an integer parameter $v.\mathit{quality}$. We assume that the slots are indexed from 1 in non-decreasing order of qualities, breaking ties arbitrarily. We allow an arbitrary number of slots that is less than the number of jobs. We also modify the edge weights so that $w(u, v)$ between job u and slot v becomes $u.\mathit{profit} + u.\mathit{priority} \cdot v.\mathit{quality}$. While we have not been able to solve the MWM problem in such a graph faster than quadratic time yet, we describe in Sect. 4 how to compute the VCG prices given an MWM of such a graph that represents a unit-demand auction instance.

Here we describe a special case of the graph structure that is introduced in the previous paragraph. Suppose that the qualities of the slots form a non-decreasing sequence which is the concatenation of two arithmetic sequences. We are able to solve the MWM problem in such a graph instance, thus we solve Problem 2 introduced in Sect. 1 in $O(n \log n)$ time. The key idea is to utilize the incremental computation of the acceptance orders so that we can find the weights of the MWMCMs between the slots whose qualities form the first arithmetic sequence (the slots before the common due date) and every possible prefix of jobs. Then we do the same between the slots whose qualities form the second arithmetic sequence (the slots after the common due date) and every possible suffix of jobs. Then in linear time we find an optimal matching by determining which jobs to assign to the first group of slots and which jobs to the second group. The details are explained in App. B.

4 Unit-Demand Auctions and VCG Prices

In this section, we view an instance $G = (U, V, w)$ of the general complete bipartite graph family introduced in Sect. 3 from the perspective of unit-demand auctions. We refer to elements of U as bids and to elements of V as items. For any bid u and item v, the weight $w(u, v)$ represents the amount offered by bid u to item v. We present an $O(n \log n)$-time algorithm for computing the VCG prices given a VCG allocation (an MWM of G).

We review some standard definitions related to unit-demand auctions and we present the details of the algorithm in App. C. Here we briefly describe the approach we take in order to obtain the desired performance. One characterization of the VCG prices is that it is the minimum stable price vector [13]. Thus a naive algorithm would start with zero prices and then look for and eliminate the instabilities. While inspecting a particular instability, the algorithm would increase the prices just enough to eliminate that instability.

We take a similar approach that uses additional care. We start with a minimum price vector that does not cause an instability involving unassigned bids, by utilizing the geometric concept of the upper envelope. We then inspect the instabilities in a particular order, with two scans of the items, first in increasing and then in decreasing order of qualities. The most expensive step is the computation of the upper envelope, which takes $O(n \log n)$ time.

Acknowledgments.

In the early stages of this work we had developed an $O(n \log^2 n)$-time algorithm for the problem considered in Sect. 2. The authors wish to thank Eric Price for pointing out how to improve this bound to $O(n \log n)$, and for allowing us to include this improvement in the present paper.

References

[1] A. Aggarwal, A. Barnoy, S. Khuller, D. Kravets, and B. Schieber. Efficient minimum cost matching and transportation using the quadrangle inequality. *Journal of Algorithms*, 19(1): 116–143, 1995.

[2] R. E. Burkard. Monge properties, discrete convexity and applications. *European Journal of Operational Research*, 176(1):1–14, 2007.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.

[5] G. Demange, D. Gale, and M. A. O. Sotomayor. Multi-item auctions. *The Journal of Political Economy*, pages 863–872, 1986.

[6] D. W. Engels, D. R. Karger, S. G. Kolliopoulos, S. Sengupta, R. N. Uma, and J. Wein. Techniques for scheduling with rejection. *Journal of Algorithms*, 49(1):175–191, 2003.

[7] L. Epstein, J. Noga, and G. J. Woeginger. On-line scheduling of unit time jobs with rejection: minimizing the total completion time. *Operations Research Letters*, 30(6):415–420, 2002.

[8] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–221, 1985.

[9] F. Glover. Maximum matching in a convex bipartite graph. *Naval Research Logistics Quarterly*, 14(3):313–316, 1967.

[10] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[11] G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities*. Cambridge University Press, 2nd edition, 1952.

[12] I. Katriel. Matchings in node-weighted convex bipartite graphs. *INFORMS Journal on Computing*, 20:205–211, December 2008.

[13] H. B. Leonard. Elicitation of honest preferences for the assignment of individuals to positions. *The Journal of Political Economy*, pages 461–479, 1983.

[14] W. Lipski, Jr. and F. P. Preparata. Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. *Acta Informatica*, 15:329–346, 1981.

[15] C. G. Plaxton. Vertex-weighted matching in two-directional orthogonal ray graphs. In *Algorithms and Computation*, volume 8283 of *Lecture Notes in Computer Science*, pages 524–534. Springer Berlin Heidelberg, 2013.

[16] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[17] A. E. Roth and M. A. O. Sotomayor. *Two-sided matching: A study in game-theoretic modeling and analysis*, volume 18. Cambridge University Press, 1992.

[18] D. Shabtay, N. Gaspar, and L. Yedidsion. A bicriteria approach to scheduling a single machine with job rejection and positional penalties. *Journal of Combinatorial Optimization*, 23 (4):395–424, 2012.

[19] D. Shabtay, N. Gaspar, and M. Kaspi. A survey on offline scheduling with rejection. *Journal of Scheduling*, 16(1):3–28, 2013.

[20] L. S. Shapley and M. Shubik. The assignment game I: The core. *International Journal of Game Theory*, 1(1):111–130, 1971.

[21] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, July 1985.

[22] S. A. Slotnick. Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research*, 212(1):1–11, 2011.

[23] S. A. Slotnick and T. E. Morton. Selecting jobs for a heavily loaded shop with lateness penalties. *Computers and Operations Research*, 23(2):131–140, 1996.

[24] G. Steiner and J. S. Yeomans. A linear time algorithm for determining maximum matchings in convex, bipartite graphs. *Computers and Mathematics with Applications*, 31:91–96, 1996.

A An Augmented Binary Search Tree Algorithm

In this appendix, the proofs that are omitted in Sect. 2 and an implementation of the algorithm that is outlined in Sect. 2.3 are presented. First, App. A.1 proves Lemmas 2, 3, and 4. Then, App A.2 gives the details of the augmentation of the BST described in Sect. 2.3 and proves Theorem 1. Finally, App. A.3 presents an implementation of the algorithm as described in the proof of Theorem 1.

A.1 Proofs of Lemmas 2, 3, and 4

The following lemma is used in the proof of Lemma 2.

Lemma 7. Let $G = (U, V, E)$ be an edge-weighted bipartite graph, let V' be a subset of V, let G' be the subgraph of G induced by the set of vertices $U \cup V'$, and let u be a vertex in U that is essential for G'. Then u is essential for G.

Proof. Assume that the claim is false, and let M be an MWMCM of G such that u is not matched in M. Let M' be an MWMCM of G'. Since u is essential for G', vertex u is matched in M'. Let G'' denote the graph $(U, V, M \oplus M')$. Since u is matched in M' and not in M, we deduce that u has degree one in G''. Since no vertex in G'' has degree more than two, we conclude that the connected component of G'' that includes u is a path, call it P, and that u is an endpoint of P. The edges of P alternate between M and M'; let X denote $P \cap M$ and let X' denote $P \cap M'$.

Case 1: P is of odd length. Since u is an endpoint of P and is u is matched in M', we deduce that $|X'| = |X| + 1$. It follows that $(M \setminus X) \cup X'$ is a matching of G with cardinality one higher than that of M, contradicting our assumption that M is an MWMCM of G.

Case 2: P is of even length. Thus $|X| = |X'|$ and every vertex in V that belongs to P is matched in both M and M', and hence belongs to V'. It follows that $(M' \setminus X') \cup X$ is an MCM of G'; in what follows, we refer to this MCM of G' as M''. Let W denote the total weight of the edges in X and let W' denote the total weight of the edges in X'.

Case 2.1: $W < W'$. Thus $(M \setminus X) \cup X'$ is an MCM of G with weight higher than that of M, contradicting our assumption that M is an MWMCM of G.

Case 2.2: $W > W'$. Thus M'' is an MCM of G' with weight higher than that of M', contradicting our assumption that M' is an MWMCM of G'.

Case 2.3: $W = W'$. Thus M'' is an MWMCM of G that does not match u, contradicting our assumption that u is essential for G'. □

Proof of Lemma 2. Immediate from Lemma 7. □

The following lemma is used in the proof of Lemma 3.

Lemma 8. For any integers i and j such that $1 \leq j \leq i \leq |U|$, we have

$$best(i,j) = \begin{cases} best(i-1, j-1) + u & \text{if job } u \text{ is essential for } G_{i,j} \\ best(i-1, j) & \text{otherwise,} \end{cases}$$

where u denotes the job with index i.

Proof. Immediate from the definition of $best(i,j)$ and from Lemma 2. □

Proof of Lemma 3. For any integer i such that $0 \leq i \leq |U|$, let $P(i)$ denote the predicate "for any integer j such that $0 \leq j \leq i$, $matching(best(i,j))$ is an MWMCM of $G_{i,j}$". We prove by induction on i that $P(i)$ holds for all i such that $0 \leq i \leq |U|$. It is easy to see that $P(0)$ holds. Let i be an integer such that $0 < i \leq |U|$, and let u denote the job with index i. We now complete the proof by arguing that $P(i-1)$ implies $P(i)$. It is easy to see that $matching(best(i,0))$ is an MWMCM of $G_{i,0}$. Thus in the remainder of the argument we may assume that j is an integer such that $1 \leq j \leq i$. We consider two cases.

Case 1: Job u is essential for $G_{i,j}$. Let v denote the slot with index j. By Lemma 1, and since the index of job u is greater than that of any job in $G_{i-1,j-1}$, we can obtain an MWMCM of $G_{i,j}$ by adding the edge (u,v) to an arbitrary MWMCM of $G_{i-1,j-1}$. By the induction hypothesis, $matching(best(i-1,j-1))$ is an MWMCM of $G_{i-1,j-1}$. Thus the matching obtained by adding edge (u,v) to $matching(best(i-1,j-1))$ is an MWMCM of $G_{i,j}$. Since Lemma 8 implies that $best(i,j)$ is equal to $best(i-1,j-1)+u$, and since the index of job u is greater than that of any job in $best(i-1,j-1)$, the latter matching is equal to $matching(best(i,j))$. We conclude that $matching(best(i,j))$ is an MWMCM of $G_{i,j}$, as required.

Case 2: Job u is not essential for $G_{i,j}$. Thus any MWMCM of $G_{i-1,j}$ is an MWMCM of $G_{i,j}$. By the induction hypothesis, we conclude that $matching(best(i-1,j))$ is an MWMCM of $G_{i,j}$. Since Lemma 8 implies that $best(i,j)$ is equal to $best(i-1,j)$, we conclude that $matching(best(i,j))$ is an MWMCM of $G_{i,j}$, as required. □

Proof of Lemma 4. Every job in $best(i,j)$ has index at most i. Since $\sigma_i[j]$ belongs to $best(i,j)$, we conclude that $i' \leq i$. Let p denote the position of $\sigma_i[j]$ in $\sigma_{i'}$. By definition, job $\sigma_i[j]$ occurs in position j of σ_i. By the definition of $\sigma_{i'}$, job $\sigma_i[j]$ occurs in position p of $\sigma_{i'}$. Since $\sigma_{i'}$ is a subsequence of σ_i, we conclude that $best(i,j-1)$ consists of $better(\sigma_i[j])$ plus $j-p$ additional jobs, each with index greater than i'. Since each job in $better(\sigma_i[j])$ has index less than i', the claim follows. □

A.2 Proof of Theorem 1

In order to reason about the augmented BST algorithm of Sect. 2.3, we find it useful to define $\mathcal{T}$ as the set of all binary trees T satisfying the following two conditions: (1) the predicate $ordered(T,i)$ holds for some i; (2) each node x in T has integer fields $x.size$, $x.sum$, $x.sizeLeft$, and $x.sumLeft$ (in addition to the field $x.job$ implied by condition (1)).

For any tree T in $\mathcal{T}$ and any node x in T, we define $subtree(x)$ as the set of all jobs that are associated with the nodes in the subtree rooted at x. For convenience we define $subtree(nil)$ as the empty set.

For any tree T in $\mathcal{T}$ and any node x in T, we define the predicate $augmented(x)$ to hold if the field $x.size$ is equal to $|subtree(x)|$, and the field $x.sum$ is equal to $sum(subtree(x))$. We refer to $x.size$ and $x.sum$ together as the ***augmented fields*** of node x.

For any tree T in $\mathcal{T}$ and any node x in T, we define the predicate $historical(x)$ to hold if the field $x.sizeLeft$ is equal to $|better(x.job)|$, and the field $x.sumLeft$ is equal to $sum(better(x.job))$. We refer to $x.sizeLeft$ and $x.sumLeft$ together as the ***historical fields*** of node x.

For any tree T in $\mathcal{T}$ and any integer i, we define the predicate $represent(T, i)$ to hold if $ordered(T, i)$ holds, and for every node x in T, both $augmented(x)$ and $historical(x)$ hold.

We prove the theorem by giving the details of the algorithm that we outlined earlier in Sect. 2.3. The algorithm grows an augmented BST by iteratively inserting each of the $|U|$ jobs, in index order. Let T_i denote the BST obtained by the algorithm after i iterations, $1 \leq i \leq |U|$. We establish below that $represent(T, i)$ holds for $1 \leq i \leq |U|$. Given this claim, it is easy to verify that the historical fields of a node x remain constant after the initial creation of x. Thus the historical fields could just as easily be maintained in an array outside of the tree. In contrast, the augmented fields form an integral part of the BST data structure, and may be updated as the structure of the tree changes.

We use a red-black tree to obtain the desired time bound. Cormen et al. [3, Chapters 13 and 14] present an implementation for red-black trees that rebalances the tree in logarithmic time while maintaining the augmented fields.

First we state an observation that will be utilized in the constant time comparison of the nodes and in the computation of the historical fields. Let T be a tree in $\mathcal{T}$ such that each node x in T satisfies $augmented(x)$. Then observe that we can compute $|precede(r, T)|$ and $sum(precede(r, T))$ in constant time where r denotes the root of T. Moreover, for each non-root node x in T, $|precede(x, T)|$ and $sum(precede(x, T))$ can be computed in constant time given $|precede(y, T)|$ and $sum(precede(y, T))$ where y denotes the parent of x.

The first iteration of the algorithm creates a BST T_1 with a single node x representing the job with index 1. The two historical fields of x are each initialized to zero, and the augmented fields are initialized as follows: $x.size$ is set to 1; $x.sum$ is set to the priority of the job with index 1. It is easy to verify that the predicate $represent(T_1, 1)$ holds.

We now describe a non-first iteration i, $1 < i \leq |U|$, where we assume inductively that $represent(T_{i-1}, i-1)$ holds. Let u denote the job with index i. In order to obtain T_i from T_{i-1}, we first insert a new leaf representing the job u into T_{i-1} without changing the structure of T_{i-1} other than linking the new leaf such that the resulting BST, call it T'_{i-1}, satisfies $ordered(T'_{i-1}, i)$. Note that there is a unique such BST T'_{i-1} since there is a unique position in the tree such that the new node representing u can be inserted without changing the edges between the existing nodes. This insertion position can be found by first comparing the job u with the job represented by the root of T_{i-1} (applying Lemma 6) and descending either left or right depending on whether u precedes the root in σ_i, thus following a path that terminates at the insertion position by comparing u with the jobs represented by the nodes on that path as in a standard BST insertion operation. By the observation in the previous paragraph, it is easy to see that $|precede(x, T_{i-1})|$ and $sum(precede(x, T_{i-1}))$ can be computed in constant time for each node x on the path that the binary search follows. Thus, together with the historical fields, we have the information to apply Lemma 6 in constant time for each comparison. Let y denote the new leaf that represents the job u. Again by the observation in the previous paragraph, since the last node that we compare u with is the parent of y, $|precede(y, T'_{i-1})|$ and $sum(precede(y, T'_{i-1}))$ can also be computed in constant time. Since $ordered(T'_{i-1}, i)$ implies $better(y.job)$ is equal to $precede(y, T'_{i-1})$, the historical fields of y can be computed in constant time.

The process described above, which attaches the new leaf y representing the job with index i to T_{i-1} to obtain T'_{i-1}, takes time proportional to the depth of y in T'_{i-1}. Once y is added, we can update the augmented fields of the nodes on the path from y to the root within the same time bound so that $augmented(x)$ holds for each node x on that path. Note that the augmented fields of the other nodes are not affected by the insertion, thus $represent(T'_{i-1}, i)$ holds.

The final step at iteration i is to rebalance T'_{i-1} with the RB-INSERT-FIXUP operation [3, Chapters 13 and 14] in logarithmic time, while maintaining the augmented fields, in order to obtain $O(|U| \log |U|)$ overall running time. Let T_i denote the result of the rebalancing operation on T'_{i-1}. Since the augmented fields are maintained by the rebalancing operation, the BST T_i satisfies $represent(T_i, i)$ and the algorithm proceeds to the next iteration. Note that it is easy to argue the same performance for certain other balanced BST structures, e.g., $O(|U| \log |U|)$ amortized time for splay trees [21] where each insertion, together with the associated splay operation at the end of each iteration, takes amortized logarithmic time.

A.3 An Implementation

Algorithm 1 in Fig. 2 is a concise implementation of the algorithm as described in the proof of Theorem 1. The fields $x.size$ and $x.sum$ of a node x are the augmented fields satisfying the predicate $augmented(x)$. The fields $x.sizeLeft$ and $x.sumLeft$ of a node x are the historical fields satisfying the predicate $historical(x)$.

The outer **while** loop (lines 3–32) runs for $|U|$ iterations. Each iteration i inserts the job with index i, referenced by the variable u, into the BST, whose root is pointed by the variable x at the beginning of the iteration. The inner **while** loop (lines 8–18) performs the binary search by starting from the root and descending to the insertion position of the new leaf representing the job u. When the execution reaches line 14, the variable $sizeLeft'$ is equal to $|precede(x, T)|$ and the variable $sumLeft'$ is equal to $sum(precede(x, T))$. The variable $flag$ is set depending on the outcome of the application of Lemma 6.

Lines 19–30 attach the new leaf y to the BST and set its historical fields. The FIXUP routine, which is called at line 31, fixes the augmented fields of the nodes along the path from the new leaf to the root, rebalances the BST using the standard techniques of the associated self-balancing BST implementation, and returns the root of the BST. Thus, at the end of each iteration of the outer **while** loop, the BST stores the jobs inserted so far in their acceptance order.

B The Algorithm for Problem 2

In this appendix, we describe how to solve Problem 2, which incorporates weighted tardiness penalties with respect to a common due date into the objective, in $O(n \log n)$ time. We start with formalizing the definition of the family of graphs, which we call $\mathcal{H}$, that extends the family $\mathcal{G}$ by introducing qualities to the slots as mentioned in Sect. 3. Then we define a more restricted family, $\mathcal{H}^*$, that consists of the graphs on which we encode the instances of Problem 2 as weighted matching problems. Finally we discuss an algorithm that solves the MWMCM problem on graphs drawn from the family $\mathcal{H}^*$.

Algorithm 1 INSERT(U)

```
 1: $x \leftarrow nil$
 2: $i \leftarrow 0$
 3: while $i < |U|$ do
 4:     $i \leftarrow i + 1$
 5:     $u \leftarrow$ the job from the set $U$ with index $i$
 6:     $sizeLeft, sumLeft \leftarrow 0$
 7:     $x' \leftarrow x$
 8:     while $x' \neq nil$ do
 9:         $x \leftarrow x'$
10:         $sizeLeft' \leftarrow$ if $x.left = nil$ then $sizeLeft$ else $sizeLeft + x.left.size$
11:         $sumLeft' \leftarrow$ if $x.left = nil$ then $sumLeft$ else $sumLeft + x.left.sum$
12:         $v \leftarrow$ the slot with index $sizeLeft' + 1$
13:         $v' \leftarrow$ the slot with index $x.sizeLeft + 1$
14:         $flag \leftarrow w(u, v) \leq w(x.job, v') + sumLeft' - x.sumLeft$
15:         $x' \leftarrow$ if $flag$ then $x.right$ else $x.left$
16:         $sizeLeft \leftarrow$ if $flag$ then $sizeLeft' + 1$ else $sizeLeft$
17:         $sumLeft \leftarrow$ if $flag$ then $sumLeft' + x.job.priority$ else $sumLeft$
18:     end while
19:     $y \leftarrow$ a new tree node
20:     $y.job \leftarrow u$
21:     $y.sizeLeft \leftarrow sizeLeft$
22:     $y.sumLeft \leftarrow sumLeft$
23:     $y.parent \leftarrow x$
24:     if $x \neq nil$ then
25:         if $flag$ then
26:             $x.right \leftarrow y$
27:         else
28:             $x.left \leftarrow y$
29:         end if
30:     end if
31:     $x \leftarrow$ FIXUP($y$)
32: end while
```

Figure 2: The FIXUP routine rebalances the tree and maintains the augmented fields *size* and *sum*. For a red-black tree implementation, the FIXUP routine can be implemented as the RB-INSERT-FIXUP operation [3, Chapter 13] with the addition of setting the new node's color to red at the beginning, following the guidelines in [3, Chapter 14] for the augmentation.

We define $\mathcal{H}$ as the family of all complete edge-weighted bipartite graphs $G = (U, V, w)$ such that the following conditions hold: $|U| \geq |V|$; each job u in U has two associated integers $u.\mathit{profit}$ and $u.\mathit{priority}$; the jobs are indexed from 1 in non-decreasing order of priorities, breaking ties arbitrarily; each slot v in V has an associated integer $v.\mathit{quality}$; the slots are indexed from 1 in non-decreasing order of qualities, breaking ties arbitrarily; the weight $w(u, v)$ of the edge between a job u and a slot v is equal to $u.\mathit{profit} + u.\mathit{priority} \cdot v.\mathit{quality}$. Note that a graph $G = (U, V, w)$ in $\mathcal{H}$ admits an $O(|U|)$-space representation. Also note that an input graph $G = (U, V, w)$ to the algorithm presented in Sect. 2 can be interpreted as a graph belonging to the family $\mathcal{H}$ that has $|U|$ slots with qualities forming the arithmetic sequence $1, \ldots, |U|$. Observe that the same algorithm can also be used to find an MWMCM for the case in which the qualities form a different arithmetic sequence by scaling the priorities, setting qualities to the arithmetic sequence $1, \ldots, |U|$, and modifying the profits.

We now introduce the notion of a "splitting point", a key technical concept that underlies our algorithm. Let $G = (U, V, w)$ be a graph in $\mathcal{H}$. We define U_i, V_j, and $G_{i,j}$ in the same manner as we did for a graph in $\mathcal{G}$ in Sect. 2. For any integer i such that $1 \leq i < |U|$, we define U_{-i} as the set $U \setminus U_i$. Similarly for any integer j such that $1 \leq j < |V|$, we define V_{-j} as the set $V \setminus V_j$. For any integers i and j such that $1 \leq j \leq i < |U|$ and $j < |V|$, we define $G_{-i,-j}$ as the subgraph of G induced by the vertices $U_{-i} \cup V_{-j}$. Then it is not hard to see that for any j in the range $1 \leq j < |V|$, there exists at least one integer i, which we call a *splitting point* for j, such that the union $M_1 \cup M_2$ is an MWMCM of G where M_1 is any MWMCM of $G_{i,j}$ and M_2 is any MWMCM of $G_{-i,-j}$. Note that if i is a splitting point for j, then $|U_i| \geq |V_j|$ and $|U_{-i}| \geq |V_{-j}|$.

We encode an instance of Problem 2 as a weighted matching problem on a graph drawn from a family $\mathcal{H}^*$ that is contained in $\mathcal{H}$. We define $\mathcal{H}^*$ as the family of all graphs $G = (U, V, w)$ in $\mathcal{H}$ such that the qualities of the slots in V, when visited in index order, form a non-decreasing sequence which is the concatenation of two arithmetic sequences.

Let I be an instance of Problem 2. The instance I consists of a set of n jobs to schedule, each with a profit and a weight; a common due date d and a common deadline $\overline{d}$ where we assume that $d < \overline{d} \leq n$; and a positive constant c. We encode the instance I as a graph $G = (U, V, w)$ in $\mathcal{H}^*$ such that the following conditions hold: $|U| = n$; $|V| = \overline{d}$; each u in U represents a distinct job in I; each v in V represents a distinct time slot in which a job in I can be scheduled; for each job in I and the vertex u that represents that job, $u.\mathit{profit}$ is equal to the profit of the job and $u.\mathit{priority}$ is equal to the negated weight of the job; the qualities of the slots in V are set to form the sequence $1, \ldots, d, (c + 1)(d + 1), \ldots, (c + 1)\overline{d}$. It is easy to see by inspecting the objective of Problem 2 that the instance I of Problem 2 is equivalent to the problem of finding an MWM of a graph $G = (U, V, w)$ in $\mathcal{H}^*$ that encodes I. Analogous to the case for $\mathcal{G}$ discussed in Sect. 2, the problem of finding an MWM of a graph in $\mathcal{H}^*$ can be reduced to the MWMCM problem by adding dummy jobs.

We now describe our algorithm for computing an MWMCM of a graph $G = (U, V, w)$ in $\mathcal{H}^*$. Let j denote the index such that the qualities of both V_j and V_{-j} are arithmetic sequences. If we can find the weights of MWMCMs of each $G_{i,j}$ for $j \leq i \leq |U|$ and the weights of MWMCMs of each $G_{-i,-j}$ for $1 \leq i \leq |U| - |V| + j$ in $O(|U| \log |U|)$ total time, then it takes linear time to find a splitting point for j, and thus an MWMCM of G can be constructed in $O(|U| \log |U|)$ total time.

Our algorithm consists of two extensions to the algorithm introduced for Problem 1. Let $G' = (U, V', w)$ be a graph in $\mathcal{H}$ such that the qualities of V' form an arithmetic sequence. In the remainder of this section, we use the shorthand G'_i (resp., G'_{-i}) to denote the subgraph of G'

induced by the vertices $U_i \cup V'$ (resp., $U_{-i} \cup V'$) for any integer i such that $1 \leq i \leq |U|$. The first extension, which we discuss in the next paragraph, exploits the incremental computation of the acceptance orders performed by the algorithm introduced for Problem 1, in order to compute the weights of MWMCMs of each G'_i for $|V'| \leq i \leq |U|$ in $O(|U| \log |U|)$ total time. The second extension, which we discuss in the final paragraph, finds the weights of MWMCMs of each G'_{-i} for $1 \leq i \leq |U| - |V'|$ in $O(|U| \log |U|)$ total time by a simple reduction so that the first extension is utilized. Then, by setting G' to the subgraph of G induced by the vertices $U \cup V_j$ (resp., $U \cup V_{-j}$) as an input to the first (resp., second) extension, these two extensions are used to find the weights of the MWMCMs of each $G_{i,j}$ for $j \leq i \leq |U|$ and the weights of MWMCMs of each $G_{-i,-j}$ for $1 \leq i \leq |U| - |V| + j$ in $O(|U| \log |U|)$ total time in order to compute an MWMCM of a graph $G = (U, V, w)$ in $\mathcal{H}^*$.

First we show how to modify the algorithm introduced for Problem 1 so that we can compute the weights of MWMCMs of each subgraph G'_i for $|V'| \leq i \leq |U|$ given a graph $G' = (U, V', w)$ in $\mathcal{H}$ such that the qualities of V' form an arithmetic sequence. As a preprocessing step, we scale the priorities and modify the profits so that the instance G' is transformed such that the qualities form the arithmetic sequence $1, \ldots, |V'|$, as mentioned in the observation after the definition of $\mathcal{H}$. In what follows, let T denote the BST that the algorithm introduced for Problem 1 maintains. We modify the algorithm so that we keep at most $|V'|$ nodes in T by discarding the rightmost node when necessary, as mentioned in Sect. 2.3. Due to these deletions, $\mathit{represent}(T, i)$ no longer holds for $i > |V'|$. However, it is easy to argue that, for $i > |V'|$, the BST T in the modified algorithm contains the first $|V'|$ jobs in σ_i (i.e., $\mathit{best}(i, |V'|)$), and that these jobs occur in the same order (with respect to an inorder traversal) as in the BST in the unmodified algorithm. For any integer i such that $i \geq |V'|$, let M_i denote $\mathit{matching}(\mathit{best}(i, |V'|))$. Then Lemma 3 implies that M_i is an MWMCM of G'_i. We maintain an additional BST τ that concurrently stores the same set of jobs that are present in the main BST T, however in a different order. The keys of the nodes in τ are the indices of the corresponding jobs, thus an inorder traversal of τ yields an increasing order of indices. We implement τ as a balanced BST and augment it so that we can query for the sum of the priorities of all jobs that have indices greater than that of a given job. All the insert, delete, and query operations can be implemented in logarithmic time using standard augmentation techniques [3, Chapters 14]. We utilize those queries in order to maintain the weight of M_i at each iteration $i \geq |V'|$ in the following way. First, the weight of $M_{|V'|}$ can be computed at the end of iteration $|V'|$ via an inorder traversal of τ. Now suppose that at the end of some iteration i for $i > |V'|$, the set of jobs in T is changed by an update consisting of insertion of job u, which is the job with index i, to T (also to τ) and removal of some job u' from T (also from τ). Let τ_{i-1} denote the state of the BST τ before this update. Let j' be the index of the slot that is matched to u' in M_{i-1}. Note that j' is the rank of u' in τ_{i-1}. Let U' denote the set of jobs in τ_{i-1} with indices greater than that of u'. Since u has the highest index among the jobs that are matched in M_i, each job in U' is matched to a slot with index one lower in M_i than in M_{i-1}, and every other job that is matched in M_{i-1} except u' is matched to the same slot in M_i. Then, the weight of M_i minus the weight of M_{i-1} is equal to $u.\mathit{profit} + u.\mathit{priority} \cdot |V'| - u'.\mathit{profit} - u'.\mathit{priority} \cdot j' - \mathit{sum}(U')$. Since such an update to τ and a query for $\mathit{sum}(U')$ in τ can be performed in $O(\log |V'|)$ time, we can maintain the weight of each M_i for $i \geq |V'|$ without slowing down the algorithm asymptotically.

In order to compute the weights of MWMCMs of each subgraph G'_{-i} for $1 \leq i \leq |U| - |V'|$ given a graph $G' = (U, V', w)$ in $\mathcal{H}$ such that the qualities of V' form an arithmetic sequence, we create another instance G'' by negating both the job priorities and slot qualities, and by reindexing both the jobs and the slots in reverse orders. Then we run the algorithm described in the previous paragraph on G''. Note that the weight of the edge between the job with an index i and the slot with an index j in G'' is equal to the weight of the edge between the job with index $|U| - i + 1$ and the slot with index $|V'| - j + 1$ in G'. Thus the weight of an MWMCM of G''_i is equal to the weight of an MWMCM of $G'_{-|U|+i}$ for $|V'| \leq i < |U|$.

C Computing the VCG Prices

This appendix first presents a fast algorithm for computing the VCG prices given a VCG allocation and then proves its correctness. The input to the algorithm is a graph $G = (U, V, w)$ from the family $\mathcal{H}$ introduced in App. B, and we fix such an instance G throughout the remainder of this section. As mentioned in Sect. 4, we refer to the left vertices of G as bids and to the right vertices of G as items. The edge weight $w(u, v)$ between a bid u and an item v represents the amount offered by bid u to item v. To better relate to the algorithm, we rename the quantities associated with the elements of U so that $u.\mathit{profit}$ and $u.\mathit{priority}$ of (a job in the scheduling context) u becomes $u.\mathit{intercept}$ and $u.\mathit{slope}$ of (now a bid) u, respectively. Thus the edge weight $w(u, v)$ between a bid u and an item v is $u.\mathit{intercept} + u.\mathit{slope} \cdot v.\mathit{quality}$. A VCG allocation corresponds to an (arbitrary) MWM of G and is also given as input. We begin by reviewing some standard definitions and results that prove to be useful. Some proof details are omitted and can be found in [17, Chapter 8].

For any matching M, let $\mathit{weight}(M)$ denote the total weight of the edges in M. A *surplus vector* s assigns a real value $s(u)$ to each bid u in U. A *price vector* p assigns a real value $p(v)$ to each item v in V. An *outcome* is a triple (s, p, M) such that s is a surplus vector, p is a price vector and M is a matching of G. An outcome (s, p, M) such that $\sum_{u \in U} s(u) + \sum_{v \in V} p(v) = \mathit{weight}(M)$ is said to be *feasible*. For any feasible outcome (s, p, M), we say that the pair of vectors (s, p) and the matching M are *compatible*, and that (s, p) is a *feasible payoff*.

We say that a bid u (resp., item v) blocks an outcome (s, p, M) if $s(u) < 0$ (resp., $p(v) < 0$). We say that a bid-item pair (u, v) blocks (or, u and v form a blocking pair for) an outcome (s, p, M) if $s(u) + p(v) < w(u, v)$. A feasible outcome (s, p, M) is *stable* (or, the payoff (s, p) is *stable* with M) if no bid, item, or bid-item pair blocks the outcome. For any stable outcome (s, p, M), the following are known: M is an MWM; $s(u) + p(v) = w(u, v)$ for all (u, v) matched in M; $s(u) = 0$ for all u unmatched in M; $p(v) = 0$ for all v unmatched in M. It is also known that any MWM is compatible with any stable payoff. Thus, given the stable price vector p of a stable payoff, the corresponding utility vector s is uniquely determined by:

$$s(u) = \begin{cases} w(u, v) - p(v) & \text{if item } v \text{ is allocated to } u \text{ in } M \\ 0 & \text{if } u \text{ is left unassigned in } M, \end{cases}$$

where M is any MWM. For any stable payoff (s, p), p is a *stable price vector*.

It is known that the stable price vectors form a lattice, hence, there is a unique stable price vector that is componentwise less than or equal to any other stable price vector. This *minimum stable price vector* corresponds to the VCG prices [13].

Before presenting our algorithm, we introduce some useful definitions concerning the input graph $G = (U, V, w)$. Let M be an MWM of G and let V' be the set of all items in V that are matched in M. Then for any item v in V', we define the successor of v in M as the item v' in V'

with the smallest index that is still larger than that of v. Similarly for any item v in V', we define the predecessor of v in M as the item v' in V' with the largest index that is still smaller than that of v. Note that no successor (resp., predecessor) exists for the item in V' with the largest (resp., smallest) index.

We say that an MWM M of G is a canonical MWM if for any two matched bids u and u' in M such that the item matched to u' is the successor of the item matched to u, we have $u'.slope \geq u.slope$. To keep the presentation and the correctness argument simpler, we assume that the input M to the algorithm is a canonical MWM of G. It is easy to observe that any non-canonical MWM can be converted to a canonical one by sorting the matched bids by slopes. Throughout the remainder of this section, we fix a canonical MWM M of G.

Algorithm 2 in Fig. 3 computes the minimum stable price vector in $O(n \log n)$ time given M and G. The high-level idea is to start with the minimum prices (**for** loop at lines 4–6) so that no unmatched bid can participate in a blocking pair (Lemma 11), then to raise the prices in a particular order only when necessary while maintaining the invariant that the price vector is at most the minimum stable price vector (Lemma 10). The price increases take place in two scans of the items, first in increasing and then in decreasing order of qualities. The prices are raised so that no matched item v can form a blocking pair with a bid that is matched to the predecessor or to the successor of v (Lemma 12). The latter result implies that no matched item can form a blocking pair with a matched bid (Lemma 13). We start with some lemmas, and then we establish the correctness and argue the performance of the algorithm in Theorem 2.

The following lemma is used in the proofs of Lemmas 10 and 11.

Lemma 9. Let v be an item in V. Let U' denote the set of all bids u such that $w(u, v) \geq 0$ and u is unmatched in M. If U' is empty, then $p(v)$ is 0 upon executing line 6. If U' is nonempty, then $p(v)$ is equal to $max_{u \in U'} w(u, v)$ upon executing line 6.

Proof. By the definition of the upper envelope. □

Lemma 10. Let p^* denote the minimum stable price vector. Then for any item v, the price set by the algorithm for v at any point during the execution is at most $p^*(v)$.

Proof. Let s^* denote the surplus vector corresponding to p^*. Suppose that the claim is false, and consider the first time the price of some item v is set to a value, which we denote by $p(v)$, that is greater than $p^*(v)$. We consider two cases.

Case 1: The first occurrence is at line 5. Then Lemma 9 implies that there is an unmatched bid u such that $w(u, v) = p(v) > p^*(v)$. Since p^* is stable and since u is unmatched in M, we know $s^*(u) = 0$. But then $s^*(u) + p^*(v) < w(u, v)$, contradicting the stability of p^*.

Case 2: The first occurrence is at line 12 (resp., line 19) and we denote the predecessor (resp., successor) of v by v'. Let u' be the bid that is matched to item v' in M. Then, immediately after the price of v is set to $p(v)$, we have $w(u', v) - p(v) = w(u', v') - p(v')$. Since this is the first time that the price for an item exceeds its minimum stable price, we know that $p(v') \leq p^*(v')$, and hence

$$w(u', v) - p(v) \geq w(u', v') - p^*(v').$$

But, since $p(v) > p^*(v)$, we have

$$w(u', v) - p^*(v) > w(u', v') - p^*(v'),$$

Algorithm 2 VCG-PRICES(G, M)

```
1:  Initialize $p(v)$ to 0 for all $v \in V$
2:  $L \leftarrow$ the set of lines corresponding to the bids that are not matched in $M$ (the line corresponding to a bid $u$ has slope $u.slope$ and intercept $u.intercept$)
3:  $H \leftarrow$ the upper envelope of $L$
4:  for all $v \in V$ do
5:      $p(v) \leftarrow \max(p(v), H(v.quality))$
6:  end for
7:  $V' \leftarrow$ the set of all items in $V$ that are matched in $M$
8:  $v \leftarrow$ the item in $V'$ with the smallest index
9:  while $v \neq$ the item in $V'$ with the largest index do
10:     $v' \leftarrow$ the successor of $v$ in $M$
11:     $u \leftarrow$ the bid matched to $v$ in $M$
12:     $p(v') \leftarrow \max(p(v'), p(v) + u.slope \cdot (v'.quality - v.quality))$
13:     $v \leftarrow v'$
14: end while
15: $v' \leftarrow$ the item in $V'$ with the largest index
16: while $v' \neq$ the item in $V'$ with the smallest index do
17:     $v \leftarrow$ the predecessor of $v'$ in $M$
18:     $u' \leftarrow$ the bid matched to $v'$ in $M$
19:     $p(v) \leftarrow \max(p(v), p(v') - u'.slope \cdot (v'.quality - v.quality))$
20:     $v' \leftarrow v$
21: end while
```

Figure 3: The $O(n \log n)$-time algorithm computing the minimum stable price vector given a canonical MWM M for bipartite graph $G = (U, V, w)$.

contradicting the stability of p^* since u' is matched to v' in M.

Hence the price of an item v is never increased beyond $p^*(v)$. □

Corollary 1. Let s^* denote the utility vector corresponding to the minimum stable prices. Then for any bid u, $s(u) \geq s^*(u)$ upon termination of the algorithm.

Proof. Follows from the definition of $s(u)$ and from Lemma 10. □

Lemma 11. Let u be a bid in U that is unmatched in M, and let v be an item in V. Then $s(u) + p(v) \geq w(u, v)$ upon termination of the algorithm.

Proof. Since u is not matched in M, we have $s(u) = 0$. Then the result follows from Lemma 9 and from the fact that prices never decrease. □

Lemma 12. Let v and v' be two items in V such that both v and v' are matched and v' is the successor of v in M. Let u denote the bid that is matched to v and let u' denote the bid that is matched to v'. Then $s(u) + p(v') \geq w(u, v')$ and $s(u') + p(v) \geq w(u', v)$ upon termination of the algorithm.

Proof. First consider the claim $s(u) + p(v') \geq w(u, v')$. The first **while** loop (lines 9 through 14) iterates through the items from lowest index to highest, so it ensures that

$$\begin{aligned} p(v') &\geq p(v) + u.slope \cdot (v'.quality - v.quality) \\ &= p(v) + w(u, v') - w(u, v) \\ &= w(u, v') - s(u) \end{aligned}$$

holds upon termination of the loop. The prices never decrease, so the claim remains satisfied after any change to the price of item v'. We now show that any increase to the price of item v does not violate the claim. Note that after the first **while** loop (lines 9 through 14) is terminated, such an increase can only happen once and it occurs in line 19. If it happens, the price of v is set to:

$$\begin{aligned} p(v) &= p(v') - u'.slope \cdot (v'.quality - v.quality) \\ &= p(v') - w(u', v') + w(u', v). \end{aligned} \tag{3}$$

If this increase violates the claim, then we have

$$w(u, v) - p(v) + p(v') < w(u, v'),$$

and substituting for $p(v)$ from (3) gives

$$w(u, v) + w(u', v') < w(u, v') + w(u', v),$$

which contradicts that M is an MWM because switching the items assigned to bids u and u' results in a heavier matching.

Now consider the claim $s(u') + p(v) \geq w(u', v)$. The second **while** loop (lines 16 through 21) iterates through the items from highest index to lowest, so it ensures that

$$\begin{aligned} p(v) &\geq p(v') - u'.slope \cdot (v'.quality - v.quality) \\ &= p(v') - w(u', v') + w(u', v) \\ &= w(u', v) - s(u') \end{aligned}$$

holds upon termination of the loop. □

Lemma 13. Let u be a bid in U and let v be an item in V such that both u and v are matched in M. Then $s(u) + p(v) \geq w(u, v)$ upon termination of the algorithm.

Proof. If v is matched to u in M, then $s(u) + p(v) = w(u, v)$ by definition. Suppose that u is matched to some item v' that is not equal to v. We give the proof for the case where the index of v' is smaller than that of v; the other case is symmetric. Since the index of v' is smaller than that of v, there exists a unique sequence $\langle v_1, \ldots, v_k \rangle$ of matched items in M such that $v_1 = v'$, $v_k = v$, and v_{i+1} is the successor of v_i in M for $1 \leq i < k$. Given such a sequence, let $P(i)$ denote the inequality $s(u) + p(v_i) \geq w(u, v_i)$. We show by induction that $P(k)$ holds. The base case $P(2)$ is implied by Lemma 12 since v_2 is the successor of $v_1 = v'$. Now assume that $P(i)$ holds for some i such that $2 \leq i < k$. Let u_i denote the bid to which item v_i is matched. Since v_{i+1} is the successor of v_i, Lemma 12 implies that

$$s(u_i) + p(v_{i+1}) \geq w(u_i, v_{i+1}).$$

Adding inequality $P(i)$ to the preceding inequality we obtain

$$s(u) + p(v_i) + s(u_i) + p(v_{i+1}) \geq w(u, v_i) + w(u_i, v_{i+1}),$$

and hence

$$s(u) + p(v_{i+1}) \geq w(u, v_i) + w(u_i, v_{i+1}) - w(u_i, v_i)$$

since $p(v_i) + s(u_i) = w(u_i, v_i)$.

By the definition of $w(u, v)$, we have

$$\begin{aligned} w(u, v_i) + w(u_i, v_{i+1}) - w(u_i, v_i) &= w(u, v_i) + u_i.\mathit{slope}(v_{i+1}.\mathit{quality} - v_i.\mathit{quality}) \\ &\geq w(u, v_i) + u.\mathit{slope}(v_{i+1}.\mathit{quality} - v_i.\mathit{quality}) \\ &= w(u, v_{i+1}), \end{aligned}$$

where the second line follows since $v_{i+1}.\mathit{quality} \geq v_i.\mathit{quality}$ and the assumption that M is a canonical MWM of G implies $u_i.\mathit{slope} \geq u.\mathit{slope}$. Hence $P(i + 1)$ holds. □

Lemma 14. Let u be a bid in U that is matched in M, and let v be an item in V that is not matched in M. Then $s(u) + p(v) \geq w(u, v)$ upon termination of the algorithm.

Proof. Let v' be the item that is matched to u. Let p^* denote the minimum stable price vector and let s^* denote the corresponding surplus vector. Then $s^*(u) + p^*(v) \geq w(u, v)$. Since v is unmatched in M we have $p^*(v) = 0$. Thus, Lemma 10 and the fact that the algorithm never sets a price to a negative value together imply that $p(v) = 0$. Corollary 1 implies $s(u) \geq s^*(u)$. Hence $s(u) \geq w(u, v)$, as required. □

Theorem 2. The algorithm computes the minimum stable price vector in $O(n \log n)$ time.

Proof. Since the algorithm maintains nonnegative prices, no item blocks M. Corollary 1 implies that no bid blocks M. Lemmas 11, 13, and 14 together imply that no bid-item pair blocks M. By combining the preceding observations we deduce that the final price vector is stable. Hence Lemma 10 implies that the final price vector is the minimum stable price vector.

The most expensive step in the algorithm is the computation of the upper envelope of n lines. It is known that the lower convex hull of points and the upper envelope of lines are dual to each other [4, Section 11.4]. Since the lower convex hull can be computed in $O(n \log n)$ time [16, Theorem 3.7] and the two **while** loops take linear time, the complexity of the algorithm is $O(n \log n)$. □

D Incrementally Computing the Weights for All Prefixes of Slots

Let n denote $|U|$. Once we have computed σ_n, we know a set of jobs that forms an MWMCM of each $G_{n,j}$ for $1 \leq j \leq n$, but we are not readily given the weight $W(n, j)$ of such a matching. In this appendix, we describe how to compute the weights $W(n, 1), \ldots, W(n, n)$ incrementally in linear time by scanning through σ_n. It is straightforward to compute $W(n, 1)$. When we inspect the job at position j in σ_n (which is $\sigma_n[j]$), we can compute $W(n, j)$ in constant time from $W(n, j - 1)$ because Lemma 3 implies that we can construct an MWMCM for $G_{n,j}$, namely

$\mathit{matching}(\mathit{best}(n,j))$, from $\mathit{matching}(\mathit{best}(n,j-1))$, which is an MWMCM for $G_{n,j-1}$, by introducing $\sigma_n[j]$. Let u' denote $\sigma_n[j]$ and let i' denote the index of $\sigma_n[j]$. The change in weight caused by introducing job u' to $\mathit{matching}(\mathit{best}(n,j-1))$ in order to construct $\mathit{matching}(\mathit{best}(n,j))$ has two components: (1) $w(u',v')$ where v' denotes the slot with index $|\mathit{better}(u')|+1$, since Lemma 4 implies that job u' is matched to the slot with index $|\mathit{better}(u')|+1$ in $\mathit{matching}(\mathit{best}(n,j))$; (2) the sum of the priorities of all jobs in $\mathit{best}(n,j)$ with indices greater than that of u', since each such job is matched to a slot with index one higher in $\mathit{matching}(\mathit{best}(n,j))$ than in $\mathit{matching}(\mathit{best}(n,j-1))$ and every other job is matched to the same slot in both matchings. By Lemma 4, the latter sum is equal to $\mathit{sum}(\mathit{best}(n,j-1)) - \mathit{sum}(\mathit{better}(i'))$. Since we scan the jobs in σ_n starting from position 1, we can maintain the sum of the priorities of the jobs scanned so far, thus we know $\mathit{sum}(\mathit{best}(n,j-1))$ when we reach job u'. We already store $|\mathit{better}(u')|$ and $\mathit{sum}(\mathit{better}(i'))$ at the node representing u', thus the change in weight caused by introducing the job u' can be computed in constant time.

E NP-hardness

It is natural to consider certain other problems within the setting of Problem 1, but with the goal of optimizing various other related criteria, possibly by imposing some constraints. Shabtay et al. [18] split the scheduling objective into two criteria: the scheduling cost f, which depends on the completion times of the jobs, and the rejection cost g, which is the sum of the penalties paid for the rejected jobs. In addition to the problem of minimizing $f+g$, Shabtay et al. also analyze the following two problems: minimization of f subject to $g \leq R$, where R is a given upper bound on the rejection cost; minimization of g subject to $f \leq K$, where K is a given upper bound on the value of the scheduling criterion. As we will see, it is not difficult to show that Problem 1 becomes NP-hard if we split our criteria in the same manner and aim for optimizing one while bounding the other. In this appendix, we choose to include the details of the reductions showing the NP-hardness of the mentioned variations for the sake of completeness.

Recall that the input to Problem 1 may be viewed as a graph $G = (U, V, w)$ in $\mathcal{G}$ where the weight $w(u,v)$ of an edge between a job u in U and a slot v in V with an index j is equal to $u.\mathit{profit} + u.\mathit{priority} \cdot j$. We split the expression $w(u,v) = u.\mathit{profit} + u.\mathit{priority} \cdot j$ denoting the weight of an edge (u,v) into two summands: the first term $u.\mathit{profit}$, which we call the *profit component*; the second term $u.\mathit{priority} \cdot j$, which we call the *scheduling component*. For a given MCM M of a graph G in $\mathcal{G}$, we define $f(G,M)$ as the sum of the scheduling components of the weights of the edges in M, and we define $g(G,M)$ as the sum of the profit components of the weights of the edges in M.

Given a graph G in $\mathcal{G}$, let $\mathcal{M}_G$ denote the set of all MCMs of G. Then we define the following three problems, which are analogous to the problems mentioned above from [18].

- P1: Find a matching M in $\mathcal{M}_G$ maximizing $f(G,M) + g(G,M)$.
- P2: Find a matching M in $\mathcal{M}_G$ maximizing $f(G,M)$ subject to $g(G,M) \geq R$.
- P3: Find a matching M in $\mathcal{M}_G$ maximizing $g(G,M)$ subject to $f(G,M) \geq K$.

The algorithm we introduced for Problem 1 solves P1 in $O(n \log n)$ time, where n denotes the number of jobs in G. In this section, we show that P2 and P3 are NP-hard. We define the decision version of both P2 and P3 as follows: Given a graph G in $\mathcal{G}$ and two integers K and R, is there an MCM M of G such that $f(G, M) \geq K$ and $g(G, M) \geq R$? In what follows, we refer to this decision problem as DP.

We show the NP-hardness of P2 and P3 by reducing the partition problem, which is known to be NP-complete, to DP. The partition problem is defined as follows: Given a sequence ρ of m positive integers $\langle \rho_1, \ldots, \rho_m \rangle$ with sum $\sum_{i=1}^{m} \rho_i = 2W$, is there a subsequence of ρ with sum W? We assume $m \geq 2$ and $\rho_i \leq W$ for all $1 \leq i \leq m$.

Throughout the remainder of the section, we fix an arbitrary instance ρ of this partition problem. We now describe how to transfer ρ to an instance (G, K, R) of DP. Our description introduces a variety of symbols, all of which are fixed in value, throughout the remainder of this section.

Let m denote the size of ρ. Let W denote $\frac{1}{2} \sum_{i=1}^{m} \rho_i$. For any integer i such that $1 \leq i \leq m$, let A_i denote $-2^{i-1}W$. Note that $A_i = \sum_{j=1}^{i-1} A_j - W$. For any integer i such that $1 \leq i \leq m$, let B_i denote $3^i W$.

Let G be a graph in $\mathcal{G}$ with a set U of $2m$ jobs $\{u_i, \ldots, u_{2m}\}$, and a set V of m slots $\{v_i, \ldots, v_m\}$, and where the profits and priorities are determined as follows. For any i such that $1 \leq i \leq m$, we define the profit of job u_{2i-1} as $a_{2i-1} = A_i$, the profit of job u_{2i} as $a_{2i} = A_i - \rho_i$, the priority of job u_{2i-1} as $b_{2i-1} = B_i$, and the priority of job u_{2i} as $b_{2i} = B_i + \frac{\rho_i}{i}$. Thus, for a given MCM M of G,

$$f(G, M) = \sum_{(u_i, v_j) \in M} b_i \cdot j,$$

and

$$g(G, M) = \sum_{(u_i, v_j) \in M} a_i.$$

Let K denote $\sum_{i=1}^{m} iB_i + W$, and let R denote $\sum_{i=1}^{m} A_i - W = -2^m W$. It is straightforward to verify that (G, K, R) is a DP instance, and the transformation from ρ to (G, K, R) can be performed in polynomial time.

Lemma 15. If ρ is a positive instance of the partition problem, then (G, K, R) is a positive instance of DP.

Proof. Assume that ρ is a positive instance of the partition problem. Let S be a subsequence of ρ with sum W. We construct an MCM M of G as follows: For any i such that $1 \leq i \leq m$, if ρ_i is in S then match u_{2i} with v_i; otherwise, match u_{2i-1} with v_i. It is easy to verify that $f(G, M) = K$ and $g(G, M) = R$. □

Lemma 16. Let U' be a size-m subset of U. Then among all the MCMs of G matching the jobs U', there is a unique matching M that maximizes $f(G, M)$, and this unique M matches the jobs to the slots $v_1, \ldots, v_m$ in increasing order of indices.

Proof. Observe that $b_i > b_{i-1}$ for any i such that $1 < i \leq 2m$. Hence the priorities of the jobs in U' are distinct. Then the result follows from the rearrangement inequality [11, Section 10.2, Theorem 368]. □

The following technical lemma is used in the proof of Lemma 18.

Lemma 17. $\sum_{j=1}^{i} jb_{i+j-2} \leq iB_i$.

Proof. For any j such that $1 \le j \le m-1$, we have $(j+1)b_{2j} + jb_{2j-1} \le 2(j+1)B_j$ since

$$\begin{aligned}(j+1)b_{2j} + jb_{2j-1} &= (j+1)\left(B_j + \frac{\rho_j}{j}\right) + jB_j \\ &\le (j+1)B_j + 2\rho_j + (j+1)B_j - B_j \\ &\le 2(j+1)B_j + 2W - 3W \\ &\le 2(j+1)B_j.\end{aligned}$$

Thus

$$\begin{aligned}\sum_{j=1}^{i} jb_{i+j-2} &\le \sum_{j=1}^{i-1}(j+1)b_{2j} + jb_{2j-1} \\ &\le 2\sum_{j=1}^{i-1}(j+1)B_j \\ &\le 2iB_{i-1}\sum_{j\ge 0} 3^{-j} \\ &= 3iB_{i-1} \\ &= iB_i.\end{aligned}$$

□

Lemma 18. Let M be an MCM of G such that $f(G,M) \ge K$ and $g(G,M) \ge R$. If (G,K,R) is a positive instance of DP, then for each i such that $1 \le i \le m$, exactly one of u_{2i-1} and u_{2i} is matched in M, and it is matched to v_i.

Proof. Assume that (G,K,R) is a positive instance of DP. Let $P_1(i)$ denote the predicate "at least one of the jobs u_{2i-1} and u_{2i} is matched in M", and let $P_2(i)$ denote the predicate "at most one of the jobs u_{2i-1} and u_{2i} is matched in M". Then we claim the following.

Claim 1: If $\bigwedge_{j=i+1}^{m}(P_1(j) \wedge P_2(j))$ holds for some integer i such that $1 \le i \le m$, then $P_1(i)$ holds. It is easy to prove the claim for $i = 1$. Let i be an integer such that $1 < i \le m$ and assume that the claim does not hold. Then, $\bigwedge_{j=i+1}^{m}(P_1(j) \wedge P_2(j))$ holds and neither u_{2i-1} nor u_{2i} is matched in M. We derive an upper on $f(G,M)$ as follows. Let U' denote the set of jobs that are matched in M. We know that U' consists of exactly one job from each pair (u_{2j-1}, u_{2j}) for $i < j \le m$, and i other jobs having indices less than $2i-1$. Lemma 16 implies that the unique MCM M' that matches U' and that maximizes $f(G,M')$ has the following structure: for all $i < j \le m$, the job from the pair (u_{2j-1}, u_{2j}) that is present in U' is matched to the slot v_j; the remaining i jobs in U' are assigned to the slots $v_1, \ldots, v_i$ in increasing order of indices. Let M^* denote this unique MCM, thus $f(G,M) \le f(G,M^*)$. We construct another matching M'' by assigning u_{2j} to v_j for $i < j \le m$, and by assigning $u_{i-1}, \ldots, u_{2i-2}$ to $v_1, \ldots, v_i$. An upper bound on $f(G,M'')$ is $\sum_{j=1}^{i} jb_{i+j-2} + \sum_{j=i+1}^{m} jB_j + 2W$, where the first term comes from the subset of M'' involving $v_1, \ldots, v_i$, and the rest is an upper bound for the subset of M'' involving $v_{i+1}, \ldots, v_m$. It is easy to see that $f(G,M^*) \le f(G,M'')$ since for each slot v, either both M'' and M^* match the same job to v, or the job that M'' matches to v has a priority greater than that of the job that M^* matches to v. Thus

$$\begin{aligned}
f(G,M) &\le f(G,M'') \\
&\le \sum_{j=1}^{i} jb_{i+j-2} + \sum_{j=i+1}^{m} jB_j + 2W \\
&= \sum_{j=1}^{i} jb_{i+j-2} + K - \sum_{j=1}^{i} jB_j + W \\
&\le iB_i + K - \sum_{j=1}^{i} jB_j + W \\
&= K - \sum_{j=1}^{i-1} jB_j + W \\
&< K,
\end{aligned}$$

where the fourth line follows from Lemma 17. This contradicts $f(G,M) \ge K$.

Claim 2: If $\bigwedge_{j=i+1}^{m} (P_1(j) \wedge P_2(j))$ holds for some integer i such that $1 \le i \le m$, then $P_2(i)$ holds. It is easy to prove the claim for $i = 1$. Let i be an integer such that $1 < i \le m$ and assume that the claim does not hold. Then, $\bigwedge_{j=i+1}^{m} (P_1(j) \wedge P_2(j))$ holds and both u_{2i-1} and u_{2i} are matched in M. Then, $g(G,M)$ is at most $2A_i - \rho_i + \sum_{j=i+1}^{m} A_j$. Using the equality $A_i = \sum_{j=1}^{i-1} A_j - W$, we deduce that $g(G,M)$ is at most $\sum_{j=1}^{m} A_j - W - \rho_i$, contradicting $g(G,M) \ge R$ since ρ_i is positive.

Claim 3: For each integer i such that $1 \le i \le m$, exactly one job from the pair (u_{2i-1}, u_{2i}) is matched in M. This claim is easily seen to hold by reverse induction on i using Claims 1 and 2.

Let U' denote the set of jobs that are matched in M. Lemma 16 and Claim 3 imply that the unique MCM M' that matches U' and that maximizes $f(G,M')$ matches exactly one job from each pair (u_{2i-1}, u_{2i}) to v_i for $1 \le i \le m$. Let M^* denote this unique MCM. It is easy to argue that the maximum $f(G,M')$ a matching M' that matches U' can attain is K. Since $f(G,M)$ is at least this maximum, M is M^*. □

Lemma 19. If (G,K,R) is a positive instance of DP, then ρ is a positive instance of the partition problem.

Proof. Assume that (G,K,R) is a positive instance of DP. Let M be an MCM of G such that $f(G,M) \ge K$ and $g(G,M) \ge R$. We construct a subsequence S of ρ as follows. We iterate over the slots in G from lowest index to the highest. Lemma 18 implies that a slot v_i is matched either to u_{2i-1} or to u_{2i} in M. We include ρ_i in the subsequence S if and only if v_i is matched to u_{2i} in M. Let $\sum_S$ denote the sum of the integers in the subsequence S. Then it is easy to verify that $f(G,M) = \sum_{i=1}^{m} iB_i + \sum_S$ and $g(G,M) = \sum_{i=1}^{m} A_i - \sum_S$. Finally, $f(G,M) \ge K$ implies that $\sum_S \ge W$, and $g(G,M) \ge R$ implies that $\sum_S \le W$. Hence $\sum_S = W$. □

Theorem 3. The optimization problems P2 and P3 are NP-hard.

Proof. Immediate from Lemmas 15 and 19, since DP is the decision version of both P2 and P3. □