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

Nevzat Onur Domaniç and C. Gregory Plaxton

Department of Computer Science
University of Texas at Austin
{onur,plaxton}@cs.utexas.edu

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.

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:

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

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

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 [20] 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 [16, 19]. Epstein et al. [7] focus on unit jobs but consider only the online version of the problem. Shabtay et al. [17] 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 quasilinear 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, 21]. 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 [4, 13, 18]. 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. Section 3 describes how to extend the algorithm to solve Problem 2 within the same time bound. Section 4 views the problem from a unit-demand auction perspective and briefly presents the approach we take in the $O(n \log n)$-time algorithm for computing the VCG prices. Due to space limitations, some details are omitted from this conference version. The companion technical report [5] includes all of the material in the present version plus five appendices. Some of the proofs related to the algorithm for Problem 1 and a brief implementation are deferred to App. A [5]. The details of the extension for Problem 2 are explained in App. B [5]. Appendix C [5] presents the algorithm for computing the VCG prices in detail. Finally, App. E [5] 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 *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 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 $0 \le j \le i \le |U|$, a subset $best(i, j)$ 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 $1 \le j \le i \le |U|$. 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 $1 \le i \le |U|$. 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 the companion technical report [5, Appendix 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 $0 \le i \le |U|$ 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 $1 \le p \le i$ since u is trivially essential for $G_{i,i}$. Furthermore, σ_{i-1} is a subsequence of σ_i for $1 \le i \le |U|$, 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 $1 \le j \le i \le |U|$ 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 $0 \le j \le i \le |U|$, 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 $1 \le j \le i \le |U|$, as claimed above.

Lemma 3. *Let i and j be any integers such that $0 \le j \le i \le |U|$. 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 the companion technical report [5, Appendix 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 $\mathit{better}(u)$ as the set of jobs that precede u in σ_i where i denotes the index of u. Thus $|\mathit{better}(u)| = p - 1$ where p is the position of u in σ_i. The set $\mathit{better}(u)$ is the set of jobs that precede u both in index order and in acceptance order.

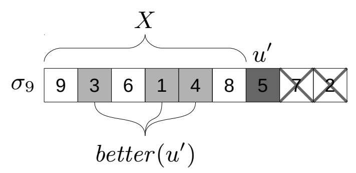
Lemma 4. *Let i and j be integers such that $1 \leq j \leq i \leq |U|$, and let i' denote the index of job $\sigma_i[j]$. Then the set of jobs in $\mathit{best}(i, j-1)$ with indices less than i' is equal to $\mathit{better}(\sigma_i[j])$.*

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

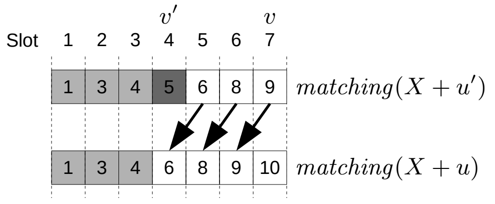
Now we are ready to discuss the idea behind the efficient computation of the acceptance orders incrementally. Assume that we already know the acceptance order σ_{i-1} of the set of the first $i-1$ jobs for some integer i such that $1 < i \leq |U|$. Let u denote the job with index i. If we can determine 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 $w(u, v) + W(i-1, j-1)$.



(a) Acceptance order σ_9



(b) The two matchings to compare

Fig. 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.

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 $matching(X + u)$ is greater than the weight of $matching(X + u')$.

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 $matching(X + u')$ and $matching(X + u)$ 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 $matching(X + u)$ and $matching(X + u')$. By Lemma 4, those jobs are the ones in the set $better(u')$, which are the jobs with indices 1, 3 and 4 in the example. Hence job u' occurs in position $|better(u')| + 1$ when we sort the set of jobs $X + u'$ by index and thus it is matched to the slot with index $|better(u')| + 1$ in $matching(X + u')$. Moreover, each job in X with index greater than that of job u' is matched to a slot with index one lower in $matching(X + u)$ than in $matching(X + u')$, as depicted by the arrows in Fig. 1b for the jobs with indices 6, 8, and 9.

Hence the weight of $matching(X + u)$ minus the weight of $matching(X + u')$ is equal to $w(u, v) - w(u', v')$ plus the sum of the priorities of all jobs in $best(i - 1, j - 1)$ with indices greater than that of u', where v' denotes the slot with index $|better(u')| + 1$. By Lemma 4, the latter sum is equal to $sum(best(i - 1, j - 1)) - sum(better(u'))$. 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 $matching(best(i - 1, j - 1) + u)$ is greater than the weight of $matching(best(i - 1, j - 1) + u')$; and (4) $w(u, v) > w(u', v') + sum(best(i -$*

$1, j-1)) - sum(better(u'))$ where v denotes the slot with index j and v' denotes the slot with 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. □

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 the companion technical report [5, Appendices 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. The companion technical report [5, 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 the companion technical report [5, Appendix 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 the companion technical report [5, Appendix 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. Aggarwal, A., Barnoy, A., Khuller, S., Kravets, D., Schieber, B.: Efficient minimum cost matching and transportation using the quadrangle inequality. Journal of Algorithms 19(1), 116–143 (1995)
2. Burkard, R.E.: Monge properties, discrete convexity and applications. European Journal of Operational Research 176(1), 1–14 (2007)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. The MIT Press, 3rd edn. (2009)
4. Demange, G., Gale, D., Sotomayor, M.A.O.: Multi-item auctions. The Journal of Political Economy pp. 863–872 (1986)

5. Domaniç, N.O., Plaxton, C.G.: Scheduling unit jobs with a common deadline to minimize the sum of weighted completion times and rejection penalties. Tech. Rep. TR–14–11, Department of Computer Science, University of Texas at Austin (September 2014)
6. Engels, D.W., Karger, D.R., Kolliopoulos, S.G., Sengupta, S., Uma, R.N., Wein, J.: Techniques for scheduling with rejection. Journal of Algorithms 49(1), 175–191 (2003)
7. Epstein, L., Noga, J., Woeginger, G.J.: On-line scheduling of unit time jobs with rejection: minimizing the total completion time. Operations Research Letters 30(6), 415–420 (2002)
8. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. Journal of Computer and System Sciences 30(2), 209–221 (1985)
9. Glover, F.: Maximum matching in a convex bipartite graph. Naval Research Logistics Quarterly 14(3), 313–316 (1967)
10. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics 5, 287–326 (1979)
11. Hardy, G.H., Littlewood, J.E., Pólya, G.: Inequalities. Cambridge University Press, 2nd edn. (1952)
12. Katriel, I.: Matchings in node-weighted convex bipartite graphs. INFORMS Journal on Computing 20, 205–211 (December 2008)
13. Leonard, H.B.: Elicitation of honest preferences for the assignment of individuals to positions. The Journal of Political Economy pp. 461–479 (1983)
14. Lipski, Jr., W., Preparata, F.P.: Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. Acta Informatica 15, 329–346 (1981)
15. Plaxton, C.G.: Vertex-weighted matching in two-directional orthogonal ray graphs. In: Algorithms and Computation, Lecture Notes in Computer Science, vol. 8283, pp. 524–534. Springer Berlin Heidelberg (2013)
16. Shabtay, D., Gaspar, N., Kaspi, M.: A survey on offline scheduling with rejection. Journal of Scheduling 16(1), 3–28 (2013)
17. Shabtay, D., Gaspar, N., Yedidsion, L.: A bicriteria approach to scheduling a single machine with job rejection and positional penalties. Journal of Combinatorial Optimization 23(4), 395–424 (2012)
18. Shapley, L.S., Shubik, M.: The assignment game I: The core. International Journal of Game Theory 1(1), 111–130 (1971)
19. Slotnick, S.A.: Order acceptance and scheduling: A taxonomy and review. European Journal of Operational Research 212(1), 1–11 (2011)
20. Slotnick, S.A., Morton, T.E.: Selecting jobs for a heavily loaded shop with lateness penalties. Computers and Operations Research 23(2), 131–140 (1996)
21. Steiner, G., Yeomans, J.S.: A linear time algorithm for determining maximum matchings in convex, bipartite graphs. Computers and Mathematics with Applications 31, 91–96 (1996)