Bipartite Matching with Linear Edge Weights*

Nevzat Onur Domaniç[1], Chi-Kit Lam[2], and C. Gregory Plaxton[3]

1 **University of Texas at Austin, Dept. of Computer Science, Austin, TX, USA**
onur@cs.utexas.edu
2 **University of Texas at Austin, Dept. of Computer Science, Austin, TX, USA**
geocklam@cs.utexas.edu
3 **University of Texas at Austin, Dept. of Computer Science, Austin, TX, USA**
plaxton@cs.utexas.edu

Abstract

Consider a complete weighted bipartite graph G in which each left vertex u has two real numbers *intercept* and *slope*, each right vertex v has a real number *quality*, and the weight of any edge (u, v) is defined as the intercept of u plus the slope of u times the quality of v. Let m (resp., n) denote the number of left (resp., right) vertices, and assume that $m \geq n$. We develop a fast algorithm for computing a maximum weight matching (MWM) of such a graph. Our algorithm begins by computing an MWM of the subgraph induced by the n right vertices and an arbitrary subset of n left vertices; this step is straightforward to perform in $O(n \log n)$ time. The remaining $m - n$ left vertices are then inserted into the graph one at a time, in arbitrary order. As each left vertex is inserted, the MWM is updated. It is relatively straightforward to process each such insertion in $O(n)$ time; our main technical contribution is to improve this time bound to $O(\sqrt{n} \log^2 n)$. This result has an application related to unit-demand auctions. It is well known that the VCG mechanism yields a suitable solution (allocation and prices) for any unit-demand auction. The graph G may be viewed as encoding a special kind of unit-demand auction in which each left vertex u represents a unit-demand bid, each right vertex v represents an item, and the weight of an edge (u, v) represents the offer of bid u on item v. In this context, our fast insertion algorithm immediately provides an $O(\sqrt{n} \log^2 n)$-time algorithm for updating a VCG allocation when a new bid is received. We show how to generalize the insertion algorithm to update (an efficient representation of) the VCG prices within the same time bound.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Weighted bipartite matching, Unit-demand auctions, VCG allocation and pricing

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.28

1 Introduction

Given an undirected graph $G = (V, E)$, a *matching* of G is a subset M of E such that no two edges in M share an endpoint. If G is a weighted graph, we define the weight of a matching as the sum of the weights of its constituent edges. The problem of finding a maximum weight matching (MWM) of a weighted bipartite graph, also known as the "assignment problem" in operations research, is a basic and well-studied problem in combinatorial optimization. A classic algorithm for the assignment problem is the Hungarian method [11], which admits an $O(|V|^3)$-time implementation. For dense graphs with arbitrary edge weights, this time bound remains the fastest known. Fredman and Tarjan [5] introduce Fibonacci heaps,

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

and by utilizing this data structure to speed up shortest path computations, they obtain a running time of $O(|V|^2 \log |V| + |E| \cdot |V|)$ for the maximum weight bipartite matching problem. When the edge weights are integers in $\{0, \ldots, N\}$, Duan and Su [4] give a scaling algorithm with running time $O(|E|\sqrt{|V|} \log N)$. In this paper, we consider a restricted class of complete weighted bipartite graphs where the edge weights have a special structure. Both unweighted and weighted matching problems in restricted classes of bipartite graphs have been studied extensively. Glover [7], Lipski and Preparata [13], Gabow and Tarjan [6], Steiner and Yeomans [15], and Katriel [10] study matching problems in convex bipartite graphs, the graphs in which the right vertices can be ordered in such a way that the neighbors of each left vertex are consecutive. Plaxton [14] studies vertex-weighted matchings in two-directional orthogonal ray graphs, which generalize convex bipartite graphs.

In the present paper, we develop a fast algorithm for computing an MWM of a complete weighted bipartite graph with the following special structure: there are m left vertices, each of which has two associated real values, a "slope" and an "intercept"; there are n right vertices, each of which has an associated real "quality"; for each left vertex u and right vertex v, the weight of edge (u, v) is given by the slope of u times the quality of v plus the intercept of u. Since the weight of any edge (u, v) is determined by evaluating the linear function specified by u (via the slope and intercept) on the quality of v, we refer to this problem as *bipartite matching with linear edge weights*. Assuming that $m \geq n$, we solve this problem in $O(m\sqrt{n} \log^2 n)$ time. We begin by solving the problem on a subgraph induced by the n right vertices and an arbitrary subset of n left vertices; this turns out to be easy to accomplish in $O(n \log n)$ time via sorting. We then insert the remaining left vertices one at a time, in arbitrary order. As each left vertex is inserted, we update the solution in $O(\sqrt{n} \log^2 n)$ time. It is relatively straightforward to process each such insertion in $O(n)$ time, yielding an overall $O(mn)$ time bound. Our algorithm provides a significant improvement over the latter bound, which is the fastest previous result that we are aware of.

In recent work that is closely related to the current paper, Domaniç and Plaxton [3] present a fast algorithm for bipartite matching with linear edge weights in the special case where the qualities of the right vertices form an arithmetic sequence. Assuming that $m \geq n$, their algorithm runs in $O(m \log m)$ time. Applying that algorithm to the scheduling domain directly solves the problem of scheduling unit jobs on a single machine with a common deadline where 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. Domaniç and Plaxton [3] also provide an extension that preserves the $O(m \log m)$ time bound for the special case where the qualities correspond to the concatenation of two arithmetic sequences. This extension solves a more general scheduling problem that incorporates weighted tardiness penalties with respect to a common due date into the objective.

By removing the technical restrictions on the qualities imposed in [3], the algorithm of the present paper supports a richer edge weight structure, while continuing to admit a compact graph representation that uses space linear in the number of vertices. In terms of scheduling, the present algorithm addresses a broader class of problems than [3]; for example, it can handle symmetric earliness and tardiness penalties with respect to a common due date, and allows certain time slots to be marked as unavailable. Below we discuss another motivation for the present work, which is based on its connection to unit-demand auctions.

In a unit-demand auction of a collection of items, each bidder submits a bid that specifies a separate offer on each item, which may or may not be equal to the private valuation that the bidder has for that item [1]. The outcome of the unit-demand auction is a pricing of

the items and an allocation of each bidder to at most one item. In mechanism design, it is known that the VCG mechanism is the only mechanism for unit-demand auctions that achieves the desired properties of being efficient, strategyproof, and envy-free [8, 12]. Such an auction can be modeled as a 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 on item v. Then, a VCG allocation corresponds to an MWM of such bipartite graph, and the VCG prices correspond to the dual variables computed by the Hungarian method, i.e., they correspond to the prices having the minimum sum among the ones that are the solutions to the dual of the linear program that solves the assignment problem encoding the auction.

The main motivation for our interest in the problem we consider in this paper, given the aforementioned desirable properties of the VCG mechanism, is to find frameworks to encode unit-demand auctions that are expressive enough to have suitable applications while being restrictive enough to yield efficient algorithms for finding VCG outcomes. For instance, consider a unit-demand auction for last-minute vacation packages in which some trusted third party (e.g., TripAdvisor) assigns a "quality" rating for each package and each bidder formulates a unit-demand bid for every package by simply declaring a linear function of the qualities of packages, i.e., determining the intercept and slope of this linear function. Within this context, we can formulate an auction as a complete weighted bipartite graph in the family that we consider in this paper. In some of the popular auction sites, e.g., eBay, bidding takes place in multiple rounds. eBay implements a variant of an English auction to sell a single item; the bids are sealed, but the second highest bid (plus one small bid increment), which is the amount that the winner pays, is displayed throughout the auction. We employ a similar approach by accepting the bids one-by-one and by maintaining an efficient representation of the tentative outcome for the enlarged set of bids. We show that we can process each bid in $\tilde{O}(\sqrt{n})$ time where n denotes the number of items in the auction. More precisely, we present a data structure that is initialized by the entire set of n items; the bids are introduced one-by-one in any order; the data structure maintains a compact representation of a VCG outcome (allocation and prices) for the bids introduced so far and for the entire set of items; it takes $O(\sqrt{n}\log^2 n)$ time to introduce a bid; it takes $O(n)$ time to print the outcome at any time.

Organization. In Sect. 2, we give the formal definition of the problem and introduce some useful definitions. In Sect. 3, we present an incremental framework for solving the problem. In Sect. 4, we present a basic algorithm within the framework of Sect. 3. Built on the concepts introduced in Sect. 4, we introduce a data structure and present our fast algorithm in Sect. 5. The companion technical report [2] includes all of the material in the present version plus some details and the proofs of all lemmas and theorems, which are omitted due to space limitations. In [2, Section 6], we extend the incremental framework to compute the VCG prices, and we present the algorithm within that framework.

2 Preliminaries

A *bid* is a triple $u = (\mathit{slope}, \mathit{intercept}, \mathit{id})$ where *slope* and *intercept* are real numbers, and *id* is an integer. We use the notation $u.\mathit{slope}$ and $u.\mathit{intercept}$ to refer to the first and second components of a bid u, respectively. The bids are ordered lexicographically. An *item* is a pair $v = (\mathit{quality}, \mathit{id})$ where *quality* is a real number and *id* is an integer. We use the notation $v.\mathit{quality}$ to refer to the first component of an item v. The items are ordered lexicographically.

For any bid u and any item v, we define $w(u, v)$ as $u.intercept + u.slope \cdot v.quality$.

For any set of bids U and any set of items V, we define the pair (U, V) as a *unit-demand auction with linear edge weights* (*UDALEW*). Such an auction represents a unit-demand auction instance where the set of bids is U, the set of items is V, and each bid u in U offers an amount $w(u, v)$ on each item v in V.

A UDALEW $A = (U, V)$ corresponds to a complete weighted bipartite graph G where left vertices are U, right vertices are V, and the weight of the edge between a left vertex u and a right vertex v is equal to $w(u, v)$. Hence, for a UDALEW, we use the standard graph theoretic terminology, alluding to the corresponding graph. The family of all such graphs G corresponds to the general graph family introduced in [3].

A *matching* of a UDALEW (U, V) is a set M of bid-item pairs where each bid (resp., item) in M belongs to U (resp., V) and no bid (resp., item) appears more than once in M. The *weight* of a matching M, denoted $w(M)$, is defined as the sum, over all bid-item pairs (u, v) in M, of $w(u, v)$.

In this paper, we solve the problem of finding a VCG outcome (allocation and prices) for a given UDALEW A; a VCG allocation is any MWM of A, and we characterize the VCG prices in [2, Sect. 6.2]. We reduce the problem of finding an MWM to the problem of finding a maximum weight maximum cardinality matching (MWMCM) as follows: we enlarge the given UDALEW instance $A = (U, V)$ by adding $|V|$ dummy bids to U, each with intercept zero and slope zero; we compute an MWMCM M of the resulting UDALEW A'; we remove from M all bid-item pairs involving dummy bids.

We conclude this section with some definitions that prove to be useful in the remainder of the paper. For any totally ordered set S — such as a set of bids, a set of items, or an ordered matching which we introduce below — we make the following definitions: any integer i is an *index* in S if $1 \leq i \leq |S|$; for any element e in S, we define the *index of* e *in* S, denoted $index(e, S)$, as the position of e in the ascending order of elements in S, where the index of the first (resp., last) element, also called the *leftmost* (resp., *rightmost*) element, is 1 (resp., $|S|$); $S[i]$ denotes the element with index i in S; for any two indices i and j in S such that $i \leq j$, $S[i : j]$ denotes the set $\{S[i], \ldots, S[j]\}$ of size $j - i + 1$; for any two integers i and j such that $i > j$, $S[i : j]$ denotes the empty set; for any integer i, $S[: i]$ (resp., $S[i :]$) denotes $S[1 : i]$ (resp., $S[i : |S|]$); a subset S' is a *contiguous* subset of S if $S' = S[i : j]$ for some $1 \leq i \leq j \leq |S|$.

For any matching M, we define $bids(M)$ (resp., $items(M)$) as the set of bids (resp., items) that participate in M. A matching M is *ordered* if M is equal to $\bigcup_{1 \leq i \leq |M|} \{(U[i], V[i])\}$ where U denotes $bids(M)$ and V denotes $items(M)$. The order of the pairs in an ordered matching is determined by the order of the bids (equivalently, items) of those pairs.

3 Incremental Framework

In this section, we present an incremental framework for the problem of finding an MWMCM of a given UDALEW $A = (U, V)$. As discussed below, it is a straightforward problem if $|U| \leq |V|$. Thus, the primary focus is on the case where $|U| > |V|$. We start with a useful definition and a simple lemma.

For any set of bids U and any set of items V such that $|U| = |V|$, we define $matching(U, V)$ as the ordered matching $\{(U[1], V[1]), \ldots, (U[|U|], V[|U|])\}$.

Lemma 1 below shows how to compute an MWMCM of a UDALEW where the number of bids is equal to the number of items. The proof follows from the rearrangement inequality [9, Section 10.2, Theorem 368].

▶ **Lemma 1.** *For any UDALEW $A = (U, V)$ such that $|U| = |V|$, matching(U, V) is an MWMCM of A.*

▶ **Corollary 2.** *For any UDALEW $A = (U, V)$ such that $|U| \geq |V|$, there exists an ordered MWMCM of A.*

If $|U| < |V|$ in a given UDALEW (U, V), then it is straightforward to reduce the problem to the case where $|U| = |V|$. Let U' (resp., U'') denote the set of the bids in U having negative (resp., nonnegative) slopes. Then we find an MWMCM M' of the UDALEW $(U', V[:|U'|])$ and an MWMCM M'' of the UDALEW $(U'', V[|V| - |U''| + 1:])$, and we combine M' and M'' to obtain an MWMCM of (U, V).

It remains to consider the problem of finding an MWMCM of a UDALEW (U, V) where $|U| > |V|$. The following is a useful lemma. The proof is straightforward by an augmenting path argument; see [3, Lemma 7] for the proof of a similar claim.

▶ **Lemma 3.** *Let $A = (U, V)$ be a UDALEW such that $|U| \geq |V|$. Let u be a bid that does not belong to U. Let M be an MWMCM of A and let U' denote bids(M). Then, any MWMCM of the UDALEW $(U' + u, V)$ is an MWMCM of the UDALEW $(U + u, V)$.*

Lemma 3 shows that the problem of finding an MWMCM of a UDALEW (U, V) where $|U| = |V| + k$ reduces to k instances of the problem of finding an MWMCM of a UDALEW where the number of bids exceeds the number of items by one. Below we establish an efficient incremental framework for solving the MWMCM problem based on this reduction.

For any ordered matching M and any bid u that does not belong to $bids(M)$, we define $insert(M, u)$ as the ordered MWMCM M' of the UDALEW $A = (bids(M) + u, items(M))$ such that the bid that is left unassigned by M', i.e., $(bids(M) + u) \setminus bids(M')$, is maximum, where the existence of M' is implied by Corollary 2.

We want to devise a data structure that maintains a dynamic ordered matching M. When the data structure is initialized, it is given an ordered matching M', and M is set to M'; we say that the data structure has initialization cost $T(n)$ if initialization takes at most $T(|M'|)$ steps. Subsequently, the following two operations are supported: the *bid insertion* operation takes as input a bid u not in $bids(M)$, and transforms the data structure so that M becomes $insert(M, u)$; the *dump* operation returns a list representation of M. We say that the data structure has bid insertion (resp., dump) cost $T(n)$ if bid insertion (resp., dump) takes at most $T(|M|)$ steps.

▶ **Lemma 4.** *Let $\mathcal{D}$ be an ordered matching data structure with initialization cost $f(n)$, bid insertion cost $g(n)$, and dump cost $h(n)$. Let A be a UDALEW (U, V) such that $|U| \geq |V|$. Then an MWMCM of A can be computed in $O(f(|V|) + (|U| - |V|) \cdot g(|V|) + h(|V|))$ time.*

In Sect. 4, we give a simple linear-time bid insertion algorithm assuming an array representation of the ordered matching. Building on the concepts introduced in Sect. 4, Sect. 5 develops an ordered matching data structure with initialization cost $O(n \log^2 n)$, bid insertion cost $O(\sqrt{n} \log^2 n)$, and dump cost $O(n)$ (Theorem 8). The results of Sect. 5, together with Lemma 4, yield the $O(m\sqrt{n} \log^2 n)$ MWMCM time bound claimed in Sect. 1.

Looking from an auction perspective, as discussed in Sect. 2, our goal is to compute a VCG allocation and pricing given a UDALEW. In [2, Sect. 6], we show how to extend the data structure of Sect. 5 to maintain the VCG prices as each bid is inserted. The asymptotic time complexity of the operations remain the same; the additional computation for maintaining the VCG prices takes $O(\sqrt{n})$ time at each bid insertion, where n denotes the size of the matching maintained by the data structure.

4 A Basic Bid Insertion Algorithm

In this section, we describe a linear-time implementation of $insert(M, u)$ given an array representation of the ordered matching. The algorithm described here is not only useful because it introduces the concepts that the fast algorithm we introduce in Sect. 5 is built on, but also the same approach is used in certain "block scan" computations of that fast algorithm. We first introduce two functions that, in a sense evident by their definitions, restrict $insert(M, u)$ into two halves, left and right, of M split by u.

For any ordered matching M and any bid u that does not belong to $bids(M)$, we define $insert_L(M, u)$ (resp., $insert_R(M, u)$) as the ordered MCM M' of the UDALEW $A = (bids(M) + u, items(M))$ of maximum weight subject to the condition that the bid that is left unassigned by M', i.e., $(bids(M) + u) \setminus bids(M')$, is less (resp., greater) than u, where the ties are broken by choosing the MCM that leaves the maximum such bid unassigned; if no such MCM exists, i.e., u is less (resp., greater) than every bid in $bids(M)$, then $insert_L(M, u)$ (resp., $insert_R(M, u)$) is defined as M.

The following lemma characterizes $insert(M, u)$ in terms of $insert_L(M, u)$ and $insert_R(M, u)$; the proof directly follows from the definitions of $insert(M, u)$, $insert_L(M, u)$, and $insert_R(M, u)$.

▶ **Lemma 5.** *Let M be a nonempty ordered matching and let u be a bid that does not belong to $bids(M)$. Let M_L denote $insert_L(M, u)$ and let M_R denote $insert_R(M, u)$. Let W denote the maximum of $w(M_L)$, $w(M)$, and $w(M_R)$. Then,*

$$insert(M, u) = \begin{cases} M_R & \text{if } w(M_R) = W \\ M & \text{if } w(M) = W > w(M_R) \\ M_L & \text{otherwise.} \end{cases}$$

We now introduce some definitions that are used in Lemma 6 below to characterize $insert_L(M, u)$ and $insert_R(M, u)$.

For any ordered matching M and any two indices i and j in M, we define M_i^j as $matching(U - U[i], V - V[j])$, where U denotes $bids(M)$ and V denotes $items(M)$.

Let M be a nonempty ordered matching, let U denote $bids(M)$, and let V denote $items(M)$. Then we define $\Delta_L(M)$ as $w(M_1^{|M|}) - w(M)$, and we define $\Delta_R(M)$ as $w(M_{|M|}^1) - w(M)$. It is straightforward to see that $\Delta_L(M[i : j])$ and $\Delta_R(M[i : j])$ can be computed for any $1 \le i \le j \le |M|$ by the recurrences

$$\Delta_L(M[k-1 : j]) = \Delta_L(M[k : j]) + w(U[k], V[k-1]) - w(U[k-1], V[k-1]) \quad \text{(L1)}$$
$$\Delta_R(M[i : k+1]) = \Delta_R(M[i : k]) + w(U[k], V[k+1]) - w(U[k+1], V[k+1]) \quad \text{(R1)}$$

with base cases $\Delta_L(M[j]) = -w(U[j], V[j])$ and $\Delta_R(M[i]) = -w(U[i], V[i])$.

Let M be a nonempty ordered matching. Letting W denote $\max_{1 \le i \le |M|} w(M_i^{|M|})$, we define $\Delta_L^*(M)$ as $W - w(M)$, and we define $loser_L(M)$ as $\max\left\{i \mid w(M_i^{|M|}) = W\right\}$. Symmetrically, letting W' denote $\max_{1 \le i \le |M|} w(M_i^1)$, we define $\Delta_R^*(M)$ as $W' - w(M)$, and we define $loser_R(M)$ as $\max\left\{i \mid w(M_i^1) = W'\right\}$. By Lemma 1 and by the definitions of $\Delta_L(M)$ and $\Delta_R(M)$, it is straightforward to see that $(\Delta_L^*(M), loser_L(M)) = \max_{1 \le i \le |M|}(\Delta_L(M[i :]), i)$ and $(\Delta_R^*(M), loser_R(M)) = \max_{1 \le i \le |M|}(\Delta_R(M[: i]), i)$ (the pairs compare lexicographically). Hence, $\Delta_L^*(M[i : j])$, $loser_L(M[i : j])$, $\Delta_R^*(M[i : j])$, and $loser_R(M[i : j])$ can be

Algorithm 1 A linear-time implementation of bid insertion. The difference of the weight of an MWMCM of the UDALEW $A = (\mathit{bids}(M) + u, \mathit{items}(M))$ and that of M is equal to δ, and the maximum bid in $\mathit{bids}(M) + u$ that is unmatched in some MWMCM of A is u^*.

Input: M is an ordered matching and u is a bid that does not belong to $\mathit{bids}(M)$.
Output: $\mathit{insert}(M, u)$.

1: Let U denote $\mathit{bids}(M)$ and let V denote $\mathit{items}(M)$
2: $C \leftarrow \{(0, u)\}$
3: $k \leftarrow \mathit{index}(u, U + u)$
4: **if** $k > 1$ **then**
5: **for** $i = k - 1$ down to 1 **do**
6: Compute $\Delta_L(M[i : k-1])$ via (L1)
7: Compute $\Delta^*_L(M[i : k-1])$ and $\mathit{loser}_L(M[i : k-1])$ via (L2)
8: **end for**
9: $C \leftarrow C + (w(u, V[k-1]) + \Delta^*_L(M[\,: k-1]), U[i])$ where $i = \mathit{loser}_L(M[\,: k-1])$
10: **end if**
11: **if** $k \leq |M|$ **then**
12: **for** $i = k$ to $|M|$ **do**
13: Compute $\Delta_R(M[k : i])$ via (R1)
14: Compute $\Delta^*_R(M[k : i])$ and $\mathit{loser}_R(M[k : i])$ via (R2)
15: **end for**
16: $C \leftarrow C + (w(u, V[k]) + \Delta^*_R(M[k :\,]), U[j])$ where $j = \mathit{loser}_R(M[k :\,]) + k - 1$
17: **end if**
18: $(\delta, u^*) \leftarrow$ the lexicographically maximum pair in C
19: **return** $\mathit{matching}(U + u - u^*, V)$

computed for any $1 \leq i \leq j \leq |M|$ by the recurrences

$$(\Delta^*_L(M[k-1 : j]), \mathit{loser}_L(M[k-1 : j])) = \max\{(\Delta^*_L(M[k : j]), \mathit{loser}_L(M[k : j]) + 1), (\Delta_L(M[k-1 : j]), 1)\} \quad \text{(L2)}$$

$$(\Delta^*_R(M[i : k+1]), \mathit{loser}_R(M[i : k+1])) = \max\{(\Delta^*_R(M[i : k]), \mathit{loser}_R(M[i : k])), (\Delta_R(M[i : k+1]), k + 2 - i)\} \quad \text{(R2)}$$

with base cases $\Delta^*_L(M[j]) = -w(U[j], V[j])$, $\Delta^*_R(M[i]) = -w(U[i], V[i])$, and $\mathit{loser}_L(M[j]) = \mathit{loser}_R(M[i]) = 1$.

▶ **Lemma 6.** *Let M be a nonempty ordered matching, let U denote $\mathit{bids}(M)$, let V denote $\mathit{items}(M)$, let u be a bid that does not belong to U, let k denote $\mathit{index}(u, U + u)$, let M_L denote $\mathit{insert}_L(M, u)$, and let M_R denote $\mathit{insert}_R(M, u)$. If $k > 1$, then M_L is equal to $M^{k-1}_i + (u, V[k-1])$ and $w(M_L) = w(M) + \Delta^*_L(M[\,: k-1]) + w(u, V[k-1])$ where i denotes $\mathit{loser}_L(M[\,: k-1])$; otherwise, $M_L = M$. If $k \leq |M|$, then M_R is equal to $M^k_j + (u, V[k])$ and $w(M_R) = w(M) + \Delta^*_R(M[k :\,]) + w(u, V[k])$ where j denotes $\mathit{loser}_R(M[k :\,]) + k - 1$; otherwise, $M_R = M$.*

Lemmas 5 and 6, together with (L1), (R1), (L2), and (R2), directly suggest a linear-time computation of $\mathit{insert}(M, u)$, as shown in Algorithm 1. If $\mathit{insert}_L(M, u)$ (resp., $\mathit{insert}_R(M, u)$) is not equal to M, then the algorithm computes the difference $w(\mathit{insert}_L(M, u)) - w(M)$ (resp., $w(\mathit{insert}_R(M, u)) - w(M)$) and adds a pair at line 9 (resp., line 16) to a set C where

the first component is this difference, and the second component is the bid in $bids(M) + u$ that is left unassigned by $insert_L(M, u)$ (resp., $insert_R(M, u)$). Then by Lemma 5, the algorithm correctly returns $insert(M, u)$ by choosing the maximum pair of C at line 18.

5 A Superblock-Based Bid Insertion Algorithm

In this section, we describe an ordered matching data structure based on the concept of a "superblock", and we show how to use this data structure to obtain a significantly faster bid insertion algorithm than that presented in Sect. 4. Before beginning our formal presentation in Sect. 5.1, we provide a high-level overview of the main ideas.

Recall that an ordered matching data structure maintains a dynamic ordered matching M. Let n denote $|M|$. We maintain a partition of the bids of M into contiguous "groups" of size $\Theta(\ell)$, where ℓ is a parameter to be optimized later. The time complexity of Alg. 1 is linear because the **for** loops starting at lines 5 and 12 process bid-item pairs in M sequentially. Our rough plan is to accelerate the computations associated with this pair of loops by proceeding group-by-group. We can process a group in constant time if we are given six "auxiliary values" that depend on the "submatching" M' of M associated with the bids in the group, namely: $\Delta_L(M')$, $\Delta_R(M')$, $\Delta_L^*(M')$, $\Delta_R^*(M')$, $loser_L(M')$, and $loser_R(M')$. The auxiliary values associated with a group can be computed in $\Theta(\ell)$ time. A natural approach is to precompute these auxiliary values when a group is created or modified, or when the set of matched items associated with the group is modified. Unfortunately, a single bid insertion can cause each bid in a contiguous interval of $\Theta(n)$ bids to have a new matched item. For example, if a bid insertion introduces a "low" bid u and deletes a "high" bid u', then each bid between u and u' gets a new matched item one position to the right of its old matched item. Since a constant fraction of the groups might need to have their auxiliary values recomputed as a result of a bid insertion, the overall time complexity remains linear.

The preceding discussion suggests that it might be useful to have an efficient way to obtain the new auxiliary values of a group of bids when the corresponding interval of matched items is shifted left or right by one position. To this end, we enhance the precomputation associated with a group of bids as follows: Instead of precomputing only the auxiliary values corresponding to the group's current matched interval of items, we precompute the auxiliary values associated with shifts of $0, \pm 1, \pm 2, \ldots, \pm\Theta(\ell)$ positions around the current matched interval. That way, unless a group of bids is modified (e.g., due to a bid being deleted or inserted) we do not need to redo the precomputation with the group until it has been shifted $\Omega(\ell)$ times. Since the enhanced precomputation computes $\Theta(\ell)$ sets of auxiliary values instead of one set, a naive implementation of the enhanced precomputation has $\Theta(\ell^2)$ time complexity, leading once again to linear worst-case time complexity for bid insertion. We obtain a faster bid insertion algorithm by showing how to perform the enhanced precomputation in $O(\ell \log^2 \ell)$ time.

Our $O(\ell \log^2 \ell)$-time algorithm for performing the enhanced precomputation forms the core of our fast bid insertion algorithm. Here we briefly mention the main techniques used to perform the enhanced precomputation efficiently; the reader is referred to [2, Sect. 5.3.1] for further details. A divide-and-conquer approach is used to compute the auxiliary values associated with the functions $loser_L$ and $loser_R$ in $O(\ell \log \ell)$ time; the correctness of this approach is based on a monotonicity result (see [2, Lemmas 8 and 9]). A convolution-based approach is used to compute the auxiliary values based on Δ_L and Δ_R in $O(\ell \log \ell)$ time (see [2, Lemma 7]). The auxiliary values based on $loser_L$ (resp., $loser_R$) are used within a divide-and-conquer framework to compute the auxiliary values based on Δ_L^* (resp., Δ_R^*); in

the associated recurrence, the overhead term is dominated by the cost of evaluating the same kind of convolution as in the computation of the auxiliary values based on Δ_L and Δ_R. As a result, the overall time complexity for computing the auxiliary values based on Δ_L^* and Δ_R^* is $O(\ell \log^2 \ell)$.

Section 5.1 introduces the concept of a "block", which is used to represent a group of bids together with a contiguous interval of items that includes all of the items matched to the group. Section 5.3 presents a block data structure. When a block data structure is "initialized" with a group of bids and an interval of items, the enhanced precomputation discussed in the preceding paragraph is performed, and the associated auxiliary values are stored in tables. A handful of "fields" associated with the block are also initialized; these fields store basic information such as the number of bids or items in the block. After initialization, the block data structure is read-only: Whenever a block needs to be altered (e.g., because a bid needs to be inserted/deleted, because the block needs to be merged with an adjacent block), we destroy the block and create a new one. The operations supported by a block may be partitioned into three categories: "queries", "lookups" and "scans". Each query runs in constant time and returns the value of a specific field. Each lookup runs in constant time and uses a table lookup to retrieve one of the precomputed auxiliary values. Each of the two linear-time scan operations (one leftgoing, one rightgoing) performs a naive emulation of one of the **for** loops of Alg. 1; in the context of a given bid insertion, such operations are only invoked on the block containing the insertion position of the new bid.

Section 5.1 defines the concept of a superblock, which is used to represent an ordered matching as a sequence of blocks. A superblock-based ordered matching data structure is introduced in Sect. 5.3, where each of the constituent blocks is represented using the block data structure alluded to in the preceding paragraph. In Sect. 5.3, we simplify the presentation by setting the parameter ℓ to $\Theta(\sqrt{n})$. For this choice of ℓ, we show that bid insertion can be performed using $O(1)$ block initializations, $O(\sqrt{n})$ block queries, $O(\sqrt{n})$ block lookups, at most two block scans, and $O(\sqrt{n})$ additional overhead, resulting in an overall time complexity of $O(\sqrt{n} \log^2 n)$. In terms of the parameters ℓ and n, the approach of Sect. 5.3 can be generalized to perform bid insertion using $O(\lceil n/\ell^2 \rceil)$ block initializations, $O(n/\ell)$ block queries, $O(n/\ell)$ block lookups, at most two block scans, and $O(n/\ell)$ additional overhead; it is easy to verify that setting ℓ to $\Theta(\sqrt{n})$ minimizes the overall time complexity.

5.1 Blocks and Superblocks

We define a *block* B as a UDALEW (U, V) where $|U| \leq |V|$. For any block $B = (U, V)$, we define $\mathit{shifts}(B)$ as $|V| - |U| + 1$. For any block $B = (U, V)$ and any integer t such that $1 \leq t \leq \mathit{shifts}(B)$, we define $\mathit{matching}(B, t)$ as $\mathit{matching}(U, V[t : t + |U| - 1])$.

Let M be a nonempty ordered matching, let U denote $\mathit{bids}(M)$, and let V denote $\mathit{items}(M)$. Let m be a positive integer, and let $\langle a_0, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$, and $\langle c_1, \ldots, c_m \rangle$ be sequences of integers such that $a_0 = 0$, $a_m = |U|$, and $1 \leq b_i \leq a_{i-1} + 1 \leq a_i \leq c_i \leq |U|$ for $1 \leq i \leq m$. Let B_i denote the block $(U[a_{i-1} + 1 : a_i], V[b_i : c_i])$ for $1 \leq i \leq m$. Then the list of blocks $S = \langle B_1, \ldots, B_m \rangle$ is a *superblock*, and we make the following additional definitions: $\mathit{matching}(S)$ denotes M; $\mathit{size}(S)$ denotes $|M|$; $\mathit{bids}(S)$ denotes U; $\mathit{items}(S)$ denotes V; $\mathit{shift}(S, i)$ and $\mathit{shift}(S, B_i)$ both denote $b_i - a_{i-1}$ for $1 \leq i \leq m$; $\mathit{sum}(S, i)$ denotes a_i for $0 \leq i \leq m$; the leftmost block B_1 and the rightmost block B_m are the *boundary blocks*, the remaining blocks $B_2, \ldots, B_{m-1}$ are the *interior blocks*. Remark: For any superblock S, $\mathit{matching}(S) = \bigcup_{1 \leq i \leq |S|} \mathit{matching}(S[i], \mathit{shift}(S, i))$.

5.2 Algorithm 2

We obtain a significantly faster bid insertion algorithm than Alg. 1 by accelerating the computations associated with the **for** loops starting at lines 5 and 12. Recall that the first loop computes $\Delta_L^*(M[:k-1])$ and $loser_L(M[:k-1])$, and the second one computes $\Delta_R^*(M[k:])$ and $loser_R(M[k:])$. These two loops process a trivial representation of M pair-by-pair using the recurrences (L1), (R1), (L2), and (R2). We start by generalizing these recurrences; these generalizations allow us to compute the aforementioned values more efficiently by looping over a superblock-based representation of the matching block-by-block, instead of pair-by-pair.

Let M denote $matching(U, V)$, and let i, j, and k be three indices in M such that $i \leq j < k$. Then the following equation generalizes (L1), and it is straightforward to prove by repeated application of (L1).

$$\Delta_L(M[i:k]) = \Delta_L(M[j+1:k]) + w(U[j+1], V[j]) + \Delta_L(M[i:j]). \tag{L1$'$}$$

We also give a generalization of (L2), where the proof follows from the definitions of Δ_L^* and $loser_L$.

$$\begin{aligned}&(\Delta_L^*(M[i:k]), loser_L(M[i:k])) = \\ &\quad \max\left\{\begin{array}{l}(\Delta_L^*(M[j+1:k]), loser_L(M[j+1:k]) + j + 1 - i), \\ (\Delta_L^*(M[i:j]) + w(U[j+1], V[j]) + \Delta_L(M[j+1:k]), loser_L(M[i:j]))\end{array}\right\}\end{aligned} \tag{L2$'$}$$

Symmetric equations generalizing (R1) and (R2) are given in [2].

We use (L1$'$) and (L2$'$) within a loop that iterates over a superblock-based representation of the matching block-by-block. In each iteration of the loop, we are able to evaluate the right-hand side of (L1$'$) and (L2$'$) in constant time because the terms involving $M[j+1:k]$ are carried over from the previous iteration, and the terms involving $M[i:j]$ are already stored in precomputed tables associated with the blocks of the superblock.

The high-level algorithm is given in Alg. 2. The input is a superblock S that represents an ordered matching, denoted M (i.e., $matching(S) = M$), and a bid u that does not belong to $bids(S)$. The output is a superblock representing $insert(M, u)$. The unique bid u^* that is unmatched in $insert(M, u)$ is identified using the block-based framework alluded to above. After identifying u^*, if $u^* \neq u$, the algorithm invokes a subroutine Swap(S, u^*, u) which, given a superblock S, a bid u^* that belongs to $bids(S)$, and a bid u that does not belong to $bids(S)$, returns a superblock that represents $matching(bids(S) + u - u^*, items(S))$. The correctness of Alg. 2 is established in [2, Lemma 6], where it is shown that Alg. 2 emulates the behavior of Alg. 1.

5.3 Fast Implementation of Algorithm 2

In this section, we first present a block data structure that precomputes the auxiliary tables mentioned in Sect. 5.2 in quasilinear time, thus allowing lines 13 and 14 of Alg. 2 to be performed in constant time. We then introduce a superblock-based ordered matching data structure that stores the blocks using the block data structure, where the sizes of the blocks are optimized to balance the cost of Swap with that of the remaining operations in Alg. 2. We present our efficient implementation of Swap, which constructs only a constant number of blocks, and analyze its time complexity in [2, Sections 5.3.3 and 5.3.4].

Let S be a superblock on which a bid insertion is performed, let B be a block in S, and let M_t denote $matching(B, t)$ for $1 \leq t \leq shifts(B)$. The algorithm may query $\Delta_L(M_t)$,

Algorithm 2 A high-level bid insertion algorithm using the superblock-based representation of an ordered matching.

Input: S is a superblock and u is a bid that does not belong to $\mathit{bids}(S)$.
Output: A superblock S' such that $\mathit{matching}(S') = \mathit{insert}(\mathit{matching}(S), u)$.

1: Let M denote $\mathit{matching}(S)$, let U denote $\mathit{bids}(S)$, and let V denote $\mathit{items}(S)$
2: Let $S[i]$ be (U_i, V_i) for $1 \leq i \leq |S|$
3: $\sigma(i) \leftarrow \mathit{sum}(S, i)$ for $0 \leq i \leq |S|$
4: $C \leftarrow \{(0, u)\}$
5: $\ell \leftarrow |\{(U', V') \mid (U', V') \in S \text{ and } U'[1] < u\}|$
6: $k \leftarrow$ **if** $\ell < 1$ **then** 1 **else** $\mathit{index}(u, U_\ell + u) + 1 + \sigma(\ell - 1)$
7: **if** $k > 1$ **then**
8: **for** $i = k - 1$ down to $\sigma(\ell - 1) + 1$ **do**
9: Compute $\Delta_L(M[i : k - 1])$ via (L1)
10: Compute $\Delta^*_L(M[i : k - 1])$ and $\mathit{loser}_L(M[i : k - 1])$ via (L2)
11: **end for**
12: **for** $i = \ell - 1$ down to 1 **do**
13: Compute $\Delta_L(M[\sigma(i - 1) + 1 : k - 1])$ via (L1′)
14: Compute $\Delta^*_L(M[\sigma(i - 1) + 1 : k - 1])$ and $\mathit{loser}_L(M[\sigma(i - 1) + 1 : k - 1])$ via (L2′)
15: **end for**
16: $C \leftarrow C + (w(u, V[k - 1]) + \Delta^*_L(M[: k - 1]), U[i])$ where $i = \mathit{loser}_L(M[: k - 1])$
17: **end if**
18: **if** $k \leq |M|$ **then**
19: Compute $\Delta^*_R(M[k :])$ and $\mathit{loser}_R(M[k :])$. See [2] for the code, which is symmetric to lines 8 through 15.
20: $C \leftarrow C + (w(u, V[k]) + \Delta^*_R(M[k :]), U[j])$ where $j = \mathit{loser}_R(M[k :]) + k - 1$
21: **end if**
22: $(\delta, u^*) \leftarrow$ the lexicographically maximum pair in C
23: **return if** $u^* \neq u$ **then** $\textsc{Swap}(S, u^*, u)$ **else** S

$\Delta_R(M_t)$, $\Delta^*_L(M_t)$, $\Delta^*_R(M_t)$, $\mathit{loser}_L(M_t)$, and $\mathit{loser}_R(M_t)$ for $t = \mathit{shift}(S, B)$. If B is part of the superblocks for a series of bid insertions, then these queries may be performed for various t values. For a fast implementation of Alg. 2, instead of individually computing these quantities at query time, we efficiently precompute them during the construction of the block and store them in the following six lists. We define $\Delta_L(B)$ as the list of size $\mathit{shifts}(B)$ such that $\Delta_L(B)[t]$ is equal to $\Delta_L(M_t)$ for $1 \leq t \leq \mathit{shifts}(B)$. We define the lists $\Delta_R(B)$, $\Delta^*_L(B)$, $\Delta^*_R(B)$, $\mathit{loser}_L(B)$, and $\mathit{loser}_R(B)$ similarly. The representation of a block $B = (U, V)$ simply maintains each of the following explicitly as an array: U, V, $\Delta_L(B)$, $\Delta_R(B)$, $\Delta^*_L(B)$, $\Delta^*_R(B)$, $\mathit{loser}_L(B)$, and $\mathit{loser}_R(B)$. In what follows, we refer to that representation as the *block data structure for* B.

The main technical contribution of this paper is that we can compute the aforementioned lists efficiently as stated in the following theorem.

▶ **Theorem 7.** *The block data structure can be constructed in* $O(|V|\,(\log \mathit{shifts}(B) + \log^2 |U|))$ *time for any block* $B = (U, V)$.

We now introduce a data structure called a *superblock-based ordered matching* (*SOM*); the formal definition is deferred to [2, Sect. 5.3.2]. A SOM represents an ordered matching

M by maintaining a superblock S such that $matching(S) = M$, where S is stored as a list of block data structures.

▶ **Theorem 8.** *The SOM has initialization cost $O(n \log^2 n)$, bid insertion cost $O(\sqrt{n} \log^2 n)$, and dump cost $O(n)$.*

Theorem 8 states the main result of our paper, and is proved in [2, Sect. 5.3.4]. Here we briefly mention key performance-related properties of the SOM, deferring the details to [2, Sect. 5.3.2]. It is easy to see that Alg. 2 does not modify the superblock, except during Swap at line 23. When Swap modifies the superblock, existing blocks are not modified; rather, some existing blocks are deleted, and some newly constructed blocks are inserted. We define the blocks in a SOM so that each block has $\Theta(\sqrt{n})$ bids and $\Theta(\sqrt{n})$ items, and so that Swap can be implemented by constructing at most a constant number of blocks, where n denotes the size of the matching represented by the SOM; we give an $O(\sqrt{n} \log^2 n)$-time implementation of Swap in [2, Sections 5.3.3 and 5.3.4].

It is possible to support constant-time queries that return the bid matched to a given item with some additional bookkeeping. Queries to find whether a bid is matched or not, and if so, to return the matched item, can be implemented in logarithmic time by performing binary search. Finally, it is possible to initialize the SOM with a matching consisting of all dummy bids, each with intercept zero and slope zero, in linear time, since all of the weights involving those bids are zero, and thus it is trivial to construct the blocks.

References

1. G. Demange, D. Gale, and M. A. O. Sotomayor. Multi-item auctions. *The Journal of Political Economy*, 94:863–872, 1986.
2. N. O. Domaniç, C.-K. Lam, and C. G. Plaxton. Bipartite matching with linear edge weights. Technical Report TR–16–15, Department of Computer Science, University of Texas at Austin, October 2016.
3. N. O. Domaniç and C. G. Plaxton. Scheduling unit jobs with a common deadline to minimize the sum of weighted completion times and rejection penalties. In *Proceedings of the 25th International Symposium on Algorithms and Computation*, pages 646–657, 2014.
4. R. Duan and H.-H. Su. A scaling algorithm for maximum weight matching in bipartite graphs. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1413–1424, 2012.
5. M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34:596–615, 1987.
6. 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*, 30:209–221, 1985.
7. F. Glover. Maximum matching in a convex bipartite graph. *Naval Research Logistics Quarterly*, 14:313–316, 1967.
8. J. Green and J.-J. Laffont. Characterization of satisfactory mechanisms for the revelation of preferences for public goods. *Econometrica*, 45:427–438, 1977.
9. G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities*. Cambridge University Press, 2nd edition, 1952.
10. I. Katriel. Matchings in node-weighted convex bipartite graphs. *INFORMS Journal on Computing*, 20:205–211, 2008.
11. H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
12. H. B. Leonard. Elicitation of honest preferences for the assignment of individuals to positions. *The Journal of Political Economy*, 91:461–479, 1983.

13 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.

14 C. G. Plaxton. Vertex-weighted matching in two-directional orthogonal ray graphs. In *Proceedings of the 24th International Symposium on Algorithms and Computation*, pages 524–534, 2013.

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