Vertex-Weighted Matching in Two-Directional Orthogonal Ray Graphs

C. Gregory Plaxton *

September 20, 2013

Abstract

Let G denote an n-vertex two-directional orthogonal ray graph. A bicolored 2D representation of G requires only $O(n)$ space, regardless of the number of edges in G. Given such a compact representation of G, and a (possibly negative) weight for each vertex, we show how to compute a maximum weight matching of G in $O(n \log^2 n)$ time. The classic problem of scheduling weighted unit tasks with release times and deadlines is a special case of this problem, and we obtain an $O(n \log n)$ time bound for this special case. As an application of our more general result, we obtain an $O(n \log^2 n)$-time algorithm for computing the VCG outcome of a sealed-bid unit-demand auction in which each item has two associated numerical parameters (e.g., third-party "quality" and "seller reliability" scores) and each bid specifies the amount an agent is willing to pay for any item meeting specified lower bound constraints with respect to these two parameters.

*Department of Computer Science, University of Texas at Austin, 2317 Speedway, Stop D9500, Austin, Texas 78712–1757. Email: plaxton@cs.utexas.edu. This research was supported by NSF Grant CCF–1217980.

1 Introduction

Certain natural classes of graphs can be represented using a constant number of words of storage for each vertex, and no additional storage for each edge, since the edges are represented implicitly: Given the representation of two vertices u and v, it is possible to determine in constant time whether there is an edge between u and v. For example, consider the class of bipartite graphs where each "left vertex" corresponds to a unit job (i.e., a job requiring one unit of processing time) with a specified integer release time and deadline, each "right vertex" corresponds to a unit-time slot on a shared resource with a specified integer timestamp, and there is an edge between left vertex u and right vertex v if and only if the timestamp of the slot associated with v lies in the interval specified by the release time and deadline of the job associated with u. Such a bipartite graph is said to be "convex".

For such "compactly representable" classes of graphs, it is interesting to revisit the complexity of fundamental graph problems. By working directly with the compact representation, we seek to outperform traditional algorithms designed for the standard adjacency list representation. In this paper, we revisit the complexity of vertex-weighted matching problems on certain compactly representable classes of bipartite graphs. All of the algorithms that we develop have time complexity that is quasilinear (i.e., within a polylogarithmic factor of linear) in the number of vertices.

Matching algorithms for convex bipartite graphs have received significant attention in the literature. In the following discussion, U denotes the set of left vertices, and V denotes the set of right vertices, of a given convex bipartite graph. Glover presented a simple greedy algorithm [8] for maximum-cardinality convex bipartite matching that admits an $O(|V| + |U| \log |U|)$-time implementation using an elementary priority queue data structure. Later, van Emde Boas used a fast priority queue to obtain an $O(|V| + |U| \log \log |U|)$-time implementation of Glover's algorithm [22]. Lipski and Preparata [10] used Tarjan's fast union-find data structure [21] to devise a different algorithm running in time $O(|U|+|V|\alpha(|V|))$, where α is a functional inverse of Ackermann's algorithm. Gabow and Tarjan [6] show that this application of union-find falls into a category admitting a linear-time implementation, thereby reducing the Lipski-Preparata time bound to $O(|U| + |V|)$. Another line of work focused on eliminating the dependence of the running time on $|V|$ [7, 16], at the expense of introducing a mild technical assumption regarding the input representation. This research culminated in the $O(|U|)$-time algorithm of Steiner and Yeomans [20].

In terms of vertex-weighted matching algorithms for convex bipartite graphs, most prior research has focused on the "left-weighted" special case in which all of the right vertices have zero weight, which corresponds to the classic problem of scheduling weighted unit jobs with release times and deadlines. (In the notation of Section 2, this corresponds to the LMWM and LMWMCM problems, which are essentially equivalent.) Dekel and Sahni [5] present a parallel algorithm for left-weighted convex bipartite matching that uses $O(|U|^2)$ processors and $O(\log^2 |U|)$ time, and which is based on a sequential algorithm with $O(|U|^2)$ complexity. Brodal et al. [2] present a data structure based on the Dekel-Sahni algorithm for the problem of maintaining a maximum cardinality matching in a dynamic convex bipartite graph. Lipski and Preparata [10] use the matroid greedy framework to develop a left-weighted convex bipartite matching algorithm with time complexity $O(|U|^2 + |U| \cdot |V|)$. Plaxton [14] discusses a similar algorithm based on the matroid greedy framework, and shows how to implement this algorithm in $O(|U| \log |U| + |V| \log^2 |V|)$ time using a data structure based on augmented trees. With additional preprocessing, and making the same technical assumption regarding the input representation as in Steiner and Yeomans [20],

Plaxton improves this bound to $O(|U| + k \log^2 k)$, where $k \leq \min\{|U|, |V|\}$ denotes the size of a maximum cardinality matching.

Katriel [9] presents an $O(|E| + |V| \log |U|)$-time algorithm for the right-weighted special case of vertex-weighted matching in convex bipartite graphs. (Here E denotes the edge set of the graph.) Katriel obtains the same time bound algorithm for the general vertex-weighted matching problem in convex bipartite graphs, under the restriction that the input graph $G = (U, V, E)$ admits a matching of size $|U|$. Since the input size is $\Theta(|U|+|V|)$, and $|E|$ could be as large as $\Theta(|U|\cdot|V|)$, these algorithms have quadratic complexity.

In this paper, we present quasilinear vertex-weighted matching algorithms for a class of bipartite graphs that properly contains the class of convex bipartite graphs. The bipartite graphs that we study admit a representation in which each vertex (left or right) has an x-value and a y-value, and there is an edge from a left vertex u to a right vertex v if and only if the x-value of u is at most the x-value of v and the y-value of u is at most the y-value of v. Such a bipartite graph is called a two-dimensional orthogonal ray graph, or 2DORG (see Section 4 for a more formal definition). It is easy to see how to represent a convex bipartite graph as a 2DORG: Using the job-slot terminology introduced earlier, we can set the x-value (resp., y-value) of each left vertex to the release time (resp., negation of the deadline) of the associated job, and we can set the x-value (resp., y-value) of each right vertex to the timestamp (resp., negation of the timestamp) of the associated slot. On the other hand, the class of 2DORGs is substantially richer than the class of convex bipartite graphs. For example, it is known that the class of 2DORGs properly contains the class of interval bigraphs, which in turn properly contains the class of convex bipartite graphs [17].

We now discuss the key techniques underlying our results. A starting point for our work is the elegant linear-time algorithm of Chang [3] for computing a maximum cardinality matching (MCM) of a chordal bipartite graph. The class of chordal bipartite graphs properly contains the class of 2DORGs [17]. Chang's algorithm runs in $O(m + n)$ time on an input graph with m edges and n vertices. Recall that in the present work we are seeking running times that are quasilinear in n. We obtain an $O(n \log n)$-time implementation of Chang's algorithm by making use of a suitably augmented binary search tree (BST). Our augmented BST data structure may be viewed as a special case of the priority search tree data structure of McCreight [11]. (For a good introduction to the topic of augmented BST data structures, see Cormen et al. [4, Chapter 14].)

Most of the technical work in our paper is geared towards leveraging the aforementioned $O(n \log n)$-time MCM algorithm for 2DORGs to obtain an $O(n \log^2 n)$-time vertex-weighted matching algorithms for 2DORGs. Here we exploit the vertex-weighted matching framework of Spencer and Mayr [19]. This is a divide-and-conquer framework for reducing vertex-weighted matching to unweighted matching. The framework is valid for general (bipartite or nonbipartite) graphs. As in the case of Chang's algorithm discussed in the previous paragraph, the original Spencer-Mayr framework is not geared towards obtaining running times that are quasilinear in the number of vertices. Rather, the original framework seeks fast running times for general graphs; these bounds depend on m and n and are not quasilinear in n, even for sparse graphs. We identify a small number of basic primitives that suffice to support the Spencer-Mayr framework, and show how to implement each of these primitives in $O(n \log n)$ time on 2DORGs. One such primitive is the $O(n \log n)$-time MCM algorithm discussed in the previous paragraph. Given a current matching, another key primitive identifies all of the vertices that can be reached from some unmatched left vertex via an alternating path of unmatched and matched edges. As in the case of our MCM algorithm for 2DORGs, our $O(n \log n)$-time 2DORG implementation of the latter primitive is based

on augmented BSTs. Once we establish that all of the primitives associated with the Spencer-Mayr framework admit $O(n \log n)$-time implementations on 2DORGs, we find that the resulting divide-and-conquer recurrence solves to give an overall running time of $O(n \log^2 n)$ for vertex-weighted matching in 2DORGs.

A practical motivation for the work of the present paper is to better understand the class of sealed-bid unit-demand auctions for which it is possible to compute a suitable outcome in time that is quasilinear in the number of vertices. In certain real-time applications of combinatorial auctions, it is crucial to employ mechanisms with low time complexity. For example, in the realm of sponsored search auctions, each search query triggers a combinatorial auction in which a (potentially large) number of bidders vie for a collection of ad slots; such an auction needs to be resolved rapidly so that the search results can be provided in a timely manner. We use our vertex-weighted matching algorithm for 2DORGs to compute a VCG allocation for a certain class of sealed-bid unit-demand auctions in $O(n \log^2 n)$ time, and we show how to compute the VCG prices in $O(n \log n)$ additional time.

The remainder of the paper is organized as follows. Section 2 provides some basic definitions and lemmas. Section 3 introduces ordered and elimination-ordered representation schemes. Section 4 presents our main result, an $O(n \log^2 n)$-time algorithm to compute a maximum weight matching of any n-vertex 2DORG. Appendix A reviews the relevant aspects of the Spencer-Mayr vertex-weighted matching framework, and adapts this framework to our setting. Appendix B presents an $O(n \log n)$-time algorithm for the special case of left-weighted matching on convex bipartite graphs. Appendix C presents several useful lemmas. Appendix D describes an $O(n \log^2 n)$-time algorithm for computing the VCG outcome of a 2DORG-related class of sealed-bid unit-demand auctions.

2 Preliminaries

A *matching* of a graph $G = (V, E)$ is a subset E' of E such that the $2|E'|$ endpoints of the edges in E' are all distinct. A *maximum cardinality matching (MCM)* of G is a matching M of G such that $|M| \geq |M'|$ for all matchings M' of G. If each edge of G has an associated weight, we define the weight of a matching M, denoted $w(M)$, as the sum of the weights of its associated edges. A *maximum weight matching (MWM)* of G is a matching M of G such that $w(M) \geq w(M')$ for all matchings M' of G. A *maximum weight MCM (MWMCM)* of G is an MCM M of G such that $w(M) \geq w(M')$ for all MCMs M' of G.

This paper addresses matching problems on vertex-weighted graphs. The vertex weights induce edge weights; we are primarily interested in the case where the weight of an edge between a vertex u and a vertex v is taken to be the sum of the weights of u and v. A matching M of a vertex-weighted graph is an *MWM* (resp., *MWMCM*) of G if it is an MWM (resp., MWMCM) of the corresponding edge-weighted graph.

A graph G is bipartite if the vertex set of G can be partitioned into two sets U and V such that every edge of G has one endpoint in U and one endpoint in V. In the present paper, we address bipartite graph problems where a particular bipartition of the vertices is specified as part of the input. Throughout the remainder of the paper, we use the term *bipartite graph* to refer to a triple (U, V, E) where U is a set of "left" vertices, V is a set of "right" vertices, and every edge in E has one endpoint in U and one endpoint in V.

The primary goal of this paper is to develop fast MWM algorithms for certain classes of vertex-weighted bipartite graphs. We analyze our algorithms in the RAM model, and we assume that each vertex weight can be represented using a constant number of machine words. It will prove to be useful to first develop fast algorithms for simpler problems in which the weights of either the left vertices, or the right vertices, are effectively zeroed out. With this in mind, we define an *LMWM* (resp., *RMWM*) of a vertex-weighted bipartite graph G as an MWM of the corresponding edge-weighted graph where the weight of an edge between a left vertex u and a right vertex v is given by the weight of u (resp., v). The terms *LMWMCM* and *RMWMCM* are defined analogously.

It is easy to see that we can compute an LMWM of a given bipartite graph $G = (U, V, E)$ by first deleting all of the negative-weight left vertices, and then computing an LMWM of the resulting bipartite graph.

Given a matching M of a bipartite graph G that is not an MCM of G, Berge's lemma [1, Theorem 1] implies the existence of a matching M' of G such that $|M'| = |M| + 1$ and the set of vertices matched in M is properly contained in the set of vertices matched in M'. Applying this idea repeatedly, we find that if M is a matching of a bipartite graph $G = (U, V, E)$, there is an MCM M' of G that matches all of the vertices matched in M. It follows that if every left vertex has nonnegative weight, then any LMWMCM is an LMWM.

Combining the observations of the two preceding paragraphs, we see that an LMWM of a given bipartite graph $G = (U, V, E)$ can be obtained by deleting all of the negative-weight left vertices, and then computing an LMWMCM of the resulting bipartite graph. Thus the LMWM and LMWMCM problems are essentially the same. In the remainder of the paper, we discuss only the LMWMCM problem. Symmetric remarks hold for the RMWM and RMWMCM problems.

Spencer and Mayr [19] attribute the following lemma, which is straightforward to prove, to Mendelsohn and Dulmage [12]; Spencer and Mayr also provide a proof.

Lemma 2.1. *Let M and M' be two MCMs of a bipartite graph $G = (U, V, E)$. Then there is an MCM of G that matches the set of left vertices matched in M to the set of right vertices matched in M'.*

Lemma 2.1 plays an important role in the Spencer-Mayr vertex-weighted matching framework discussed in Appendix A, since it yields a reduction from the problem of vertex-weighted bipartite matching to the restricted case in which only the vertices on one side of the bipartition have nonzero weight. This reduction is restated below using the terminology of the present paper.

Lemma 2.2. *Let M be an LMWMCM of a vertex-weighted bipartite graph $G = (U, V, E)$, let U' be the set of left vertices of G that are matched in M, and let G' be the subgraph of G induced by $U' \cup V$. Then any RMWMCM of G' is an MWMCM of G.*

Proof. Immediate from Lemma 2.1. □

For any class C of graphs, and any integers m and n, we let $C_{m,n}$ denote the set of all graphs in C with at most m edges and at most n vertices.

A *representation scheme* ξ for a class C of graphs specifies a set $reps(\xi, G)$ of possible representations for any given graph G in the class.

Let ξ denote a representation scheme for a class C of graphs. Scheme ξ is said to have *space complexity* at most $f(m, n)$ if, for any graph G in $C_{m,n}$, the space used by any representation in $reps(\xi, G)$ is at most $f(m, n)$. Thus, for example, the standard adjacency list representation

scheme has space complexity $O(m+n)$. Scheme ξ is said to have ***MCM complexity*** at most $f(m,n)$ if there is an $f(m,n)$-time algorithm which, given any representation in $reps(\xi, G)$ of a graph G in $C_{m,n}$, computes an MCM of G. The ***MWM*** (resp., ***LMWMCM***, ***RMWMCM***, ***MWMCM***) *complexity* of ξ is defined similarly, except that the input to the $f(m,n)$-time algorithm also specifies the vertex weights.

We say that a class C of graphs is *hereditary* if any induced subgraph of a graph in C also belongs to C. A representation scheme for a hereditary class C of graphs has *induced subgraph complexity* at most $f(m,n)$ if there is an $f(m,n)$-time algorithm which, given any representation in $reps(\xi, G)$ of a graph $G = (V, E)$ in $C_{m,n}$, and any specified subset V' of V, computes a representation in $reps(\xi, G')$ where G' denotes the subgraph of G induced by V'.

Lemma 2.3. *Let ξ denote a representation scheme for a hereditary class of bipartite graphs. If ξ has induced subgraph, LMWMCM, and RMWMCM complexity at most $f(m,n)$, then ξ has MWMCM complexity $O(f(m,n))$.*

Proof. Immediate from Lemma 2.2. □

Let ξ be a representation scheme for a class C of bipartite graphs. Scheme ξ is said to have *left-to-right search complexity* at most $f(m,n)$ if there exists an $f(m,n)$-time algorithm which, given any representation in $reps(\xi, G)$ of a graph G in $C_{m,n}$, and any matching M of G, computes the set of all vertices that are reachable from some unmatched left vertex via an alternating path of unmatched and matched edges. The *right-to-left search complexity* of ξ is defined symmetrically. The *search complexity* of ξ is at most $f(m,n)$ if the left-to-right search complexity and right-to-left search complexity of ξ are each at most $f(m,n)$.

3 Ordered Representation Schemes

An *ordering* of a bipartite graph G specifies a total order over the set of left vertices of G, and a total order over the set of right vertices of G.

A representation of a bipartite graph G is *ordered* if it specifies an ordering of G, and allows the relative order of any two left (resp., right) vertices to be determined in constant time. A representation scheme ξ for a class C of bipartite graphs is *ordered* if for every G in C, every representation in $reps(\xi, G)$ is ordered.

Let ξ be an ordered representation scheme for a hereditary class C of bipartite graphs. We say that ξ has *left-to-right delete-min complexity* $f(n)$ if there exists an $f(n)$-time algorithm A which, given a representation R in $reps(\xi, G)$ for some graph G in C with at most n vertices, and a left vertex u of G, performs the following operation, which we denote $delete\text{-}min(u)$: (1) if u has one or more incident edges, then letting v denote the least right vertex adjacent to u (with respect to the total order defined over the right vertices), and letting G' denote G with right vertex v removed, A returns v and modifies R to obtain a representation in $reps(\xi, G')$; (2) if u has no incident edges, then A returns nil and leaves R unchanged. The *right-to-left delete-min complexity* of ξ is defined symmetrically, along with the associated operation $delete\text{-}min(v)$. The *delete-min complexity* of ξ is at most $f(n)$ if the left-to-right and right-to-left delete-min complexity of ξ are each at most $f(n)$. The following lemma is straightforward to prove.

Lemma 3.1. *Let ξ be an ordered representation scheme for a hereditary class of bipartite graphs. If ξ has left-to-right (resp., right-to-left) delete-min complexity $f(n)$, then ξ has left-to-right (resp., right-to-left) search complexity $O(nf(n))$. Thus if ξ has delete-min complexity at most $f(n)$, then ξ has search complexity $O(nf(n)$.*

A bipartite graph is *chordal bipartite* if each cycle of length at least six has a chord. (Remark: A chordal bipartite graph need not be chordal because chordless cycles of length four are permitted.) The class of chordal bipartite graphs has been extensively studied, and various alternative characterizations are known. One such characterization is that a bipartite graph G is chordal bipartite if and only if G is $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$-free, which means that G admits an ordering such that for any pair of left vertices u and u' such that $u < u'$, and any pair of right vertices v and v' such that $v < v'$, if u is adjacent to v, u is adjacent to v', and u' is adjacent to v, then u' is adjacent to v'. In the present paper, we refer to such an ordering as an *elimination ordering*.

A representation of a chordal bipartite graph G is *elimination-ordered* if it specifies an elimination ordering of G. A representation scheme ξ for a class C of chordal bipartite graphs is *elimination-ordered* if for every graph G in C, each representation in $reps(\xi, G)$ is elimination-ordered.

Three lemmas related to elimination-ordered representation schemes are stated and proven in Appendix C. Lemma C.1 is based on the MCM algorithm of Chang [3]. Lemmas C.2 and C.3 are useful for dealing with negative vertex weights.

4 Two-Directional Orthogonal Ray Graphs

A bipartite graph $G = (U, V, E)$ is called an *orthogonal ray graph (ORG)* if there exists a horizontal ray (i.e., a closed half-line parallel to the x-axis) corresponding to each left vertex, and a vertical ray (i.e., a closed half-line parallel to the y-axis) corresponding to each right vertex, such that a left vertex u is adjacent to a right vertex v if and only if the two corresponding rays intersect. If all of the horizontal rays go in the same direction (e.g., to the right), and all of the vertical rays go in the same direction (e.g., down), then we say that the ORG is a *two-directional ORG (2DORG)*

Various equivalent characterizations of the class of 2DORGs are known. Shrestha, Tayu, and Ueno [17] show that a graph is a 2DORG if and only if G admits a biadjacency matrix that is $\begin{bmatrix} * & 1 \\ 1 & 0 \end{bmatrix}$-free. In the notation of the present paper, this is equivalent to saying that G admits an ordering for which, for all pairs of left vertices u and u' such that $u < u'$, and all pairs of right vertices v and v' such that $v < v'$, if u is adjacent to v' and u' is adjacent to v, then u' is adjacent to v'. Notice that such an ordering is an elimination ordering, and hence every 2DORG is chordal bipartite. On the other hand, not every chordal bipartite graph is a 2DORG [18, Lemma 3.4.11].

Soto [18, Lemma 3.4.9] notes that a bipartite graph is a 2DORG if and only if it is a bicolored 2D-graph, that is, there exists a red point in the plane for each left vertex, and a blue point in the plane for each right vertex, such that there is an edge from left vertex u to right vertex v if and only if each component of the red point associated with u is at most the corresponding component of the blue point associated with v. This characterization suggests a natural *bicolored 2D representation* of a 2DORG in which each vertex is represented by a red or blue point in the x-y plane. (The

original definition also suggests such a representation, where the point corresponding to a vertex is given by the endpoint of the corresponding ray.)

Soto [18, Lemma 3.4.1] also points out that every bicolored 2D-graph admits a *bicolored rook representation*, that is, a bicolored 2D representation satisfying the following additional constraints, where n denotes the number of vertices in the graph: (1) no two of the n points share a common x-value, or a common y-value; (2) the points are all drawn from the set $[n]^2$, where n denotes the number of vertices and $[n]$ denotes $\{i \mid 0 \leq i < n\}$. Furthermore, as shown by Soto, such a bicolored rook representation can be obtained from any bicolored 2D representation in $O(n \log n)$ time using a straightforward sorting procedure.

We now define a useful elimination-ordered representation scheme, denoted ξ^*, for the class of 2DORGs. Under scheme ξ^*, our representation of an n-vertex 2DORG G consists of a bicolored 2D representation plus two additional data structures. Like a bicolored rook representation, we require the bicolored 2D representation of G to satisfy the constraint that no two of the n points share a common x-value, or a common y-value. However, we do not require all of the points to be drawn from $[n]^2$; instead, we enforce the relaxed requirement that each coordinate is an integer that can be stored in a constant number of machine words. (Under the standard RAM model assumption, the word size is sufficiently large to store any integer in $[n]$.) Before describing the two additional data structures associated with our representation, we define an ordering of G and prove that it is an elimination ordering. We define a total order $<$ over the set of left vertices as follows: $u < u'$ if and only if $y(u') < y(u)$. We define a total order $<$ over the set of right vertices as follows: $v < v'$ if and only if $x(v) < x(v')$. The following lemma establishes that this ordering is an elimination ordering.

Lemma 4.1. *Let $G = (U, V, E)$ be a 2DORG, and let R belong to $reps(\xi^*, G)$. If $u < u'$, $v < v'$, (u, v') belongs to E, and (u', v) belongs to E, then (u', v') belongs to E.*

Proof. We need to prove that $x(u') \leq x(v')$ and $y(u') \leq y(v')$.

Since (u', v) belongs to E, we have $x(u') \leq x(v)$. Since $v < v'$, we have $x(v) < x(v')$. Hence $x(u') < x(v')$.

Since (u, v') belongs to E, we have $y(u) \leq y(v')$. Since $u < u'$, we have $y(u') < y(u)$. Hence $y(u') < y(v')$. □

We now describe the two additional data structures associated with our representation of G under scheme ξ^*. These data structures may be viewed as special cases of the priority search tree data structure of McCreight [11]. The first is a red-black tree that stores all of the left vertices in increasing order with respect to the total order $<$. This red-black tree is augmented (see [4, Chapter 14] for an introduction to augmented binary search trees) by maintaining, at each node α, an integer "*min*" field equal to the minimum, over all left vertices u stored in the subtree rooted at node α, of $x(u)$. It is straightforward to maintain the *min* field while supporting the standard dictionary operations in logarithmic time. This data structure allows us to support the $delete\text{-}min(v)$ operation in logarithmic time, where v is an arbitrary right vertex.

The second data structure is a similar red-black tree that stores all of the right vertices in increasing order with respect to the total order $<$. This red-black tree is augmented by maintaining, at each node α, an integer "*max*" field equal to the maximum, over all right vertices v stored in the subtree rooted at node α, of $y(v)$. As in the case of the first data structure, it is straightforward to maintain the *max* field while supporting the standard dictionary operations in logarithmic time.

This second data structure allows us to support the $delete\text{-}min(u)$ operation in logarithmic time, where u is an arbitrary left vertex.

Lemma 4.2. *The representation scheme ξ^* for the class of 2DORGs has space and induced subgraph complexity $O(n)$, delete-min complexity $O(\log n)$, search and MCM complexity $O(n \log n)$, and LMWMCM, RMWMCM, MWMCM, and MWM complexity $O(n \log^2 n)$.*

Proof. The $O(n)$ bound on space complexity is immediate from the definition of ξ^*. For the $O(n)$ bound on induced subgraph complexity, notice that we can form each of the two augmented red-black tree data structures associated with a specified induced subgraph as follows: (1) traverse the corresponding red-black tree for the original graph to extract the desired sorted sequence of vertices; (2) arrange this sorted sequence of vertices into a perfectly balanced red-black tree structure (e.g., the same structure as is achieved in a binary heap); (3) fill in the values of the auxiliary fields in a bottom-up manner.

As indicated in our description of ξ^*, the two augmented red-black tree structures allow us to support arbitrary $delete\text{-}min(u)$ and $delete\text{-}min(v)$ operations in logarithmic time. Thus the delete-min complexity of ξ^* is $O(\log n)$.

Lemma 3.1 implies that the search complexity of ξ^* is $O(n \log n)$, and Lemma C.1 implies that the MCM complexity of ξ^* is $O(n \log n)$.

Applying Lemma A.2 with $f(m, n) = O(n \log n)$, we find that the LMWMCM and RMWMCM complexity of ξ^* is $O(n \log^2 n)$. Applying Lemma 2.3 with $f(m, n) = O(n \log^2 n)$, we find that the MWMCM complexity of ξ^* is $O(n \log^2 n)$.

It remains to bound the MWM complexity of ξ^*. Let C denote the class of all 2DORGs, and let C' denote the class of all graphs G' of the form $extend(G, U', V')$ where $G = (U, V, E)$ belongs to C, U' is a subset of U, and V' is a subset of V. By applying Lemma C.3 with $f_0(m, n) = f_1(m, n) = O(n)$, $f_2(m, n) = f_3(m, n) = O(n \log n)$, and $f_4(n) = f_5(n) = O(\log n)$, we find that there is an elimination-ordered representation scheme with dummies ξ' for C' with space and induced subgraph complexity $O(n)$, search complexity $O(n \log n)$, and delete-min complexity $O(\log n)$. Thus, reasoning in the same manner as we did above for ξ^*, we find that ξ' has MCM complexity $O(n \log n)$, and LMWMCM, RMWMCM, and MWMCM complexity $O(n \log^2 n)$.

Lemma C.3 also implies that if we are given a representation in $reps(\xi^*, G)$ of a graph $G = (U, V, E)$ in C, a subset U' of U, and a subset V' of V, then we can compute a representation in $reps(\xi', G')$ where $G' = extend(G, U', V')$ in $O(n)$ time. Since ξ' has MWMCM complexity $O(n \log^2 n)$, Lemma A.3 implies that ξ^* has MWM complexity $O(n \log^2 n)$. □

Theorem 1. *Assume that we are given a bicolored 2D-graph representation of an n-vertex, vertex-weighted 2DORG G such that each x-value, y-value, or weight is an $O(1)$-word integer. Then an MWM of G can be computed in $O(n \log^2 n)$ time.*

Proof. Since any two $O(1)$-word integers can be compared in constant time, we can construct a representation in $reps(\xi^*, G)$ in $O(n \log n)$ time. Since representation scheme ξ^* has MWM complexity $O(n \log^2 n)$ by Lemma 4.2, the claim of the theorem follows. □

References

[1] C. Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43:842–844, 1957.

[2] G. S. Brodal, L. Georgiadis, K. A. Hansen, and I. Katriel. Dynamic matchings in convex bipartite graphs. In *Proceedings of the 32nd International Symposium on Mathematical Foundations of Computer Science*, pages 406–417, August 2007.

[3] M.-S. Chang. Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs. In *Proceedings of the 7th Annual International Symposium on Algorithms and Computation, LNCS 1178*, pages 146–155, December 1996.

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

[5] E. Dekel and S. Sahni. A parallel matching algorithm for convex bipartite graphs and applications to scheduling. *Journal of Parallel and Distributed Computing*, 1:185–205, 1984.

[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] G. Gallo. An $O(n \log n)$ algorithm for the convex bipartite matching problem. *Operations Research Letters*, 3:31–34, 1984.

[8] F. Glover. Maximum matching in convex bipartite graphs. *Naval Research Logistic Quarterly*, 14:313–316, 1967.

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

[10] 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.

[11] E. M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14:257–276, 1985.

[12] N. S. Mendelsohn and A. L. Dulmage. Some generalizations of the problem of distinct representatives. *Canadian Journal of Mathematics*, 10:230–241, 1958.

[13] N. Nisan. Introduction to mechanism design (for computer scientists). In N. Nisan, T. Roughgarden, É. Tardos, and V Vazirani, editors, *Algorithmic Game Theory*, chapter 9. Cambridge University Press, Cambridge, Massachussetts, 2007.

[14] C. G. Plaxton. Fast scheduling of weighted unit jobs with release times and deadlines. In *Proceedings of the 35th Annual International Colloquium on Automata, Languages, and Programming, LNCS 5125*, pages 222–233, July 2008.

[15] A. E. Roth and M. A. O. Sotomayor. *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*. Cambridge University Press, New York, NY, 1990.

[16] M. G. Scutellà and G. Scevola. A modification of Lipski-Preparata's algorithm for the maximum matching problem on bipartite convex graphs. *Ricerca Operativa*, 46:63–77, 1988.

[17] A. M. S. Shrestha, S. Tayu, and S. Ueno. On orthogonal ray graphs. *Discrete Applied Mathematics*, 158:1650–1659, 2010.

[18] J. A. Soto. *Contributions on Secretary Problems, Independents Sets of Rectangles and Related Problems*. PhD thesis, Department of Mathematics, Massachusetts Institute of Technology, June 2011.

[19] T. H. Spencer and E. W. Mayr. Node weighted matching. In *Proceedings of the 11th Annual International Colloquium on Automata, Languages, and Programming, LNCS 172*, pages 454–464, July 1984.

[20] 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.

[21] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22:215–225, 1975.

[22] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6:80–82, 1977.

A A Vertex-Weighted Matching Framework

Spencer and Mayr [19] present a framework for reducing vertex-weighted matching to maximum cardinality matching, in both bipartite graphs and general graphs. Their work assumes an adjacency list representation of the input graph. In this section, we adapt the bipartite graph results of Spencer and Mayr to allow for other representation schemes.

Let $G = (U, V, E)$ be a vertex-weighted bipartite graph. Fix an order of the left vertices of G from "lightest" to "heaviest" by sorting by weight, breaking ties arbitrarily. For any integer ℓ such that $0 \leq \ell \leq |U|$, let $lmwmcms(G, \ell)$ denote the set of all LMWMCMs M of G such that all of the $|U| - \ell$ heaviest left vertices of G are matched in M.

Using the notation of the present paper, we now describe the recursive routine that lies at the heart of the Spencer-Mayr vertex-weighted matching framework. This routine, call it A, takes as input a bipartite graph $G = (U, V, E)$ and an integer ℓ such that $1 \leq \ell \leq |U|$ and $lmwmcms(G, \ell)$ is nonempty, and returns an LMWMCM in $lmwmcms(G, \ell)$. Thus, by calling routine A with the parameter ℓ set to $|U|$, we obtain an LMWMCM of G. A symmetric approach can be used to obtain an RMWMCM of G.

Spencer and Mayr analyze the time complexity of routine A under the assumption that the input bipartite graph G is provided in adjacency list format. In the proof of the following lemma, we revisit the formulation of the recurrence associated with routine A in order to bound its performance with respect to other representation schemes.

Lemma A.1. *Let ξ denote a representation scheme for a hereditary class C of bipartite graphs. Let $T(m, n, \ell)$ denote the worst-case time complexity of the recursive routine A of Spencer and Mayr when the input consists of a representation in $reps(\xi, G)$ of a bipartite graph $G = (U, V, E)$ in $C_{m,n}$, and a positive integer ℓ that is at most $|U|$ (resp., $|V|$). If ξ has induced subgraph, MCM, and left-to-right (resp., right-to-left) search complexity at most $f(m, n)$, then the LMWMCM (resp., RMWMCM) complexity of ξ is at most $T(m, n, |U|)$ (resp., $T(m, n, |V|)$), where $T(m, n, 1) =$*

$O(f(m,n))$ *and for* $\ell > 1$, $T(m,n,\ell)$ *is at most the maximum, over all nonnegative integers* m_0, n_0, m_1, *and* n_1 *such that* $m_0 + m_1 \leq m$ *and* $n_0 + n_1 \leq n$, *of*

$$T(m_0, n_0, \lceil \ell/2 \rceil) + T(m_1, n_1, \lceil \ell/2 \rceil) + O(f(m,n)).$$

Proof. We address only the LMWMCM version of the lemma; a symmetric argument applies for the RMWMCM version.

By the correctness of routine A, which is established in Spencer and Mayr, and the definition of $T(m,n,\ell)$, we conclude that the LMWMCM complexity of ξ is at most $T(m,n,n)$.

Remark: In order to verify some of the claims made below concerning the operations performed within routine A, the reader may wish to consult the pseudocode given in Spencer and Mayr [19, Figure 2], where this routine is referred to as RBPMP. (The acronym RBPMP stands for "restricted bipartite positive matching problem". In our terminology, RBPMP corresponds to LMWMCM, or symmetrically, to RMWMCM.)

When $\ell = 1$, routine A computes an MCM M of G. If M matches all of U, routine A terminates and returns M. Otherwise, letting G' denote the induced subgraph of G formed by deleting from G the lightest vertex in U, routine A returns an MCM of G'. Since the induced subgraph and MCM complexity of ξ are assumed to be at most $f(m,n)$, we conclude that $T(m,n,1) = O(f(m,n))$.

When $\ell > 1$, routine A computes an MCM of G by performing two top-level recursive calls, to be discussed in greater detail below, along with a constant number of operations in each of the following categories.

1. Forming a representation in $reps(\xi, G')$ of some induced subgraph G' of G. Since the induced subgraph complexity of ξ is assumed to be at most $f(m,n)$, the cost of such an operation is at most $f(m,n)$.

2. Given a representation in $reps(\xi, G')$ of some induced subgraph G' of G, computing an MCM of G'. Since the MCM complexity of ξ is assumed to be at most $f(m,n)$, the cost of such an operation is at most $f(m,n)$.

3. Given a representation in $reps(\xi, G')$ of some induced subgraph G' of G, and an MCM M of G', computing the set of all vertices in G' that are reachable from an unmatched left vertex (with respect to M) via an alternating path of unmatched-matched edges. Since the left-to-right search complexity of ξ is assumed to be at most $f(m,n)$, the cost of such an operation is at most $f(m,n)$.

4. Operations that are easily performed in $O(n)$ time: identifying the $\lceil \ell/2 \rceil$ lightest vertices in U; taking the union of three matchings of total size at most n; determining how many of the left vertices in a given induced subgraph of G belong to the set of ℓ lightest left vertices in G.

Since $f(m,n) = \Omega(n)$, we deduce that the total cost of a constant number of operations in the above categories is $O(f(m,n))$. The two top-level recursive calls performed by routine A have arguments (G_0, ℓ_0) and (G_1, ℓ_1), respectively, where G_0 and G_1 denote disjoint induced subgraphs of G and ℓ_0 and ℓ_1 are each at most $\lceil \ell/2 \rceil$. The claimed upper bound on $T(m,n,\ell)$ for $\ell > 1$ follows. □

Lemma A.2. *Let ξ denote a representation scheme for a hereditary class of bipartite graphs with induced subgraph, MCM, and left-to-right (resp., right-to-left) search complexity at most $f(m,n)$. Assume that for all nonnegative integers m, m_0, m_1, n, n_0, n_1 such that $m_0 + m_1 \leq m$ and $n_0 + n_1 \leq n$, we have $f(m_0,n_0) + f(m_1,n_1) \leq f(m,n)$. Then the LMWMCM (resp., RMWMCM) complexity of ξ is $O(f(m,n)\log n)$.*

Proof. This lemma follows by solving the recurrence associated with Lemma A.1. The claimed solution is easily verified by induction. □

In the special case of the adjacency list representation scheme for the class of all bipartite graphs, which is the case considered by Spencer and Mayr, the preceding lemma implies LMWMCM and RMWMCM complexity $O(m \log n)$. In the applications addressed in Section 4 and Appendix B of the present paper, we use Lemma A.2 in the context of compact representation schemes for which $f(m,n)$ is quasilinear in n.

The MWM and MWMCM problems are different in the presence of negative-weight vertices. Spencer and Mayr provide a simple technique to eliminate negative weight vertices through the introduction of dummy vertices. In this paper, we will apply the same technique in the context of restricted classes of bipartite graphs. For now we introduce the technique in the context of arbitrary bipartite graphs. For any bipartite graph $G = (U,V,E)$, any subset U' of U, and any subset V' of V, let $extend(G,U',V')$ denote the graph G' obtained from G by adding $|U'| + |V'|$ "dummy"vertices, each with exactly one incident edge, as follows: for each left vertex u in U', a dummy right vertex is added with an edge to u; for each right vertex v in V', a dummy left vertex is added with an edge to v. The transformation associated with the following simple lemma is justified by Spencer and Mayr; we include a proof below.

Lemma A.3. *Let $G = (U,V,E)$ be a vertex-weighted bipartite graph, let U' (resp., V') denote the set of all vertices in U (resp., V) with negative weight. Let G' denote the vertex-weighted bipartite graph $extend(G,U',V')$ where: the weight of each vertex in $(U-U') \cup (V-V')$ is the same as in G; the weight of each vertex in $U' \cup V'$ is zero; the weight of each dummy vertex is the negation of the weight in G of its unique neighbor. Let M be an MWMCM of G'. Then $M \cap E$ is an MWM of G.*

Proof. Let us call a matching M_0 of G *good* if for every unmatched vertex u in $U - U'$ and every unmatched vertex v in $V - V'$, the edge (u,v) does not belong to E.

Let A denote the set of all MCMs of G'. Let B denote the set of all good matchings of G. For any matching M_0 in A, let $f(M_0)$ denote $M_0 \cap E$, and observe that $f(M_0)$ belongs to B. For any matching M_1 in B, let $g(M_1)$ denote the matching of G' obtained from M_1 by adding as many edges incident on dummy vertices as possible, and observe that $g(M_1)$ belongs to A. Furthermore, for any matching M_0 in A, we have $g(f(M_0)) = M_0$. Hence f defines a one-to-one correspondence from A to B.

Let the sum of the negative vertex weights in G be $-W$. Observe that for any matching M_0 in A, the weight of M_0 exceeds the weight of $f(M_0)$ by exactly W. Since f defines a one-to-one correspondence from A to B, and since M is a maximum-weight element of A, we conclude that $f(M)$ is a maximum-weight element of B. Since the weight of any vertex in $(U-U') \cup (V-V')$ is nonnegative, some MWM of G belongs to B. Thus $f(M)$ is an MWM of G, as required. □

B Convex Bipartite Graphs

A bipartite graph $G = (U, V, E)$ is *convex* if there exists a total order $<$ over the set of right vertices V such that for any left vertex u, and any right vertices v, v', and v'', if $v < v' < v''$ and u is adjacent to v and v'', then u is adjacent to v'. It is straightforward to prove that the class of convex bipartite graphs is properly contained in the class of 2DORGs. Any convex bipartite graph admits an *interval-value* representation in which each left vertex has an associated interval, each right vertex has an associated value, and a left vertex u is adjacent to a right vertex v if and only if the value associated with v belongs to the interval associated with u.

Lemma B.1. *There exists an ordered representation scheme ξ for the class of convex bipartite graphs with space, induced subgraph, left-to-right search, and MCM complexity $O(n)$, and LMWMCM complexity $O(n \log n)$. Furthermore, given an interval-value representation of a convex bipartite graph G, it is possible to compute a representation in $reps(\xi, G)$ in $O(n \log n)$ time.*

Proof. Let ξ' denote the interval-value representation scheme for the class of convex bipartite graphs. We obtain the desired ordered representation scheme ξ by sorting the left vertices by the maxima of their associate intervals, breaking ties arbitrarily, and by sorting the right vertices by their associated values. The time to convert from ξ' to ξ is clearly $O(n \log n)$. As part of this conversion, we also replace (if possible) the endpoints of each interval associated with a left vertex with values that correspond to suitable right vertices; for example, if the minimum endpoint of the interval is x, then we replace x with the minimum value of any right vertex that is at least x. This can be done in $O(n)$ time via merging. The space and induced graph complexity of ξ is clearly $O(n)$. Representation scheme ξ is suitable for applying the $O(|U| + |V|)$-time MCM algorithm discussed in Section 1; hence the MCM complexity of ξ is $O(n)$.

Just as the aforementioned MCM algorithm is based on union-find, we can use union-find to establish an $O(n)$ bound on the left-to-right search complexity of ξ. The idea is to maintain a queue of left vertices that have been "discovered" but have not yet had their outgoing edges "explored". Recall that left-to-right search is performed with respect to a given matching, call it M. Initially, the queue contains the left vertices that are unmatched in M, and the right vertices are arranged in a doubly-linked list of singleton union-find trees, in sorted order. As the search progresses, and vertices on the right are discovered, each maximal contiguous sequence of discovered right vertices is maintained as a single entry consisting of a union-find tree containing the vertices in this sequence. The search terminates when the queue of left vertices is empty. While it is nonempty, a left vertex u is removed from the queue, and find operations are performed corresponding to the endpoints of the associated interval. It is then straightforward to explore the relevant section of the linked list of union-find trees containing the right vertices to locate all as yet undiscovered right vertices lying in the interval associated with u, and to update the list of union-find trees appropriately. Whenever a right vertex v is discovered for the first time, we check whether it is matched in M. If so, the match of v is added to the queue of discovered left vertices. As in the case of the MCM algorithm discussed in Section 1, the special structure of this union-find application results in a linear time bound.

The claimed $O(n \log n)$ bound on the LMWMCM complexity of ξ follows by Lemma A.2. □

Theorem 2. *Assume that we are given an interval-value representation of an n-vertex, vertex-weighted convex bipartite graph $G = (U, V, E)$ such that each interval endpoint, value, and weight is an $O(1)$-word integer. Then an LMWMCM of G can be computed in $O(n \log n)$ time.*

Proof. Immediate from Lemma B.1. □

C Additional Lemmas

The following lemma is an easy consequence of the MCM algorithm of Chang [3]. Since this algorithm is straightforward to state and analyze, we do so below using the notations of the present paper.

Lemma C.1. *Suppose ξ is an elimination-ordered representation scheme for a hereditary class of chordal bipartite graphs. If ξ has either left-to-right or right-to-left delete-min complexity at most $f(n)$, then ξ has MCM complexity $O(nf(n))$.*

Proof. By symmetry, it suffices to consider the case where the left-to-right delete-min complexity is at most $f(n)$. Consider the following algorithm. Initialize matching M to the empty matching. Then process left vertices from lowest to highest with respect to the elimination order. To process a left vertex u, perform a $delete\text{-}min(u)$ operation. If this operation returns a right vertex v (as opposed to nil), then add edge (u, v) to M. After processing all of the left vertices, we claim that M is an MCM.

To prove the claim, let k denote the highest integer such that some MCM of G matches the k lowest left vertices in the same way as M, and let M' denote an MCM of G that matches the k lowest left vertices in the same way as M. If k is equal to the number of left vertices, then $M = M'$, and we conclude that M is an MCM, as required. Otherwise, let u denote the least left vertex that is matched differently in M' than in M. In each of the following two cases, we derive a contradiction by exhibiting an MCM M'' that matches the $k + 1$ lowest left vertices in the same way as M.

Case 1: u is unmatched in M'. Thus u is matched to some right vertex v in M. The definition of M' implies that v is matched to some left vertex u' in M' such that $u < u'$. But then we obtain a contradiction with $M'' = M' - (u', v) + (u, v)$.

Case 2: u is matched to right vertex v' in M'. Thus the $delete\text{-}min(u)$ operation performed when processing u does not return nil, since the minimum is taken over a set of right vertices that includes v', and so is nonempty. Hence this operation returns a right vertex v. Since M and M' match u differently, we conclude that $v < v'$. If v is unmatched in M', then we obtain a contradiction with $M'' = M' - (u, v') + (u, v)$. Thus v is matched to some left vertex u' in M', where $u < u'$. By the elimination ordering property, we conclude that u' is adjacent to v', which yields a contradiction with $M'' = M' - (u, v') - (u', v) + (u, v) + (u', v')$. □

The next two lemmas are used to handle negative vertex weights.

Lemma C.2. *Let G be a chordal bipartite graph, and let graph G' be obtained from G by adding a new left vertex u (resp., right vertex v) that is adjacent to exactly one right (resp., left) vertex. Then we can extend an elimination ordering of G to an elimination ordering of G' by prepending u (resp., v) to the ordered list of left (resp., right) vertices.*

Proof. Immediate from the definition of an elimination ordering. □

Below we consider representation schemes for classes of graphs that include "dummy" vertices in the sense discussed in Appendix A. We say that such a scheme is a representation scheme *with dummies*. For a representation scheme with dummies, we require the representation of a graph to encode the dummy/non-dummy status of each vertex.

Lemma C.3. *Let C be a class of chordal bipartite graphs. Let ξ be an elimination-ordered representation scheme for C with space complexity at most $f_0(m,n)$, induced subgraph complexity at most $f_1(m,n)$, left-to-right search complexity at most $f_2(m,n)$, right-to-left search complexity at most $f_3(m,n)$, left-to-right delete-min complexity at most $f_4(n)$, and right-to-left delete-min complexity at most $f_5(n)$. Let C' denote the class of all graphs G' of the form $extend(G,U',V')$ where $G=(U,V,E)$ belongs to C, U' is a subset of U, and V' is a subset of V. Then there is an elimination-ordered representation scheme with dummies ξ' for C' with space complexity $O(f_0(m,n)+n)$, induced subgraph complexity $O(f_1(m,n)+n)$, left-to-right search complexity $O(f_2(m,n)+n)$, right-to-left search complexity $O(f_3(m,n)+n)$, left-to-right delete-min complexity $O(f_4(n))$, and right-to-left delete-min complexity $O(f_5(n))$. Furthermore, given a representation in $reps(\xi,G)$ of a graph $G=(U,V,E)$ in C, a subset U' of U, and a subset V' of V, we can compute a representation in $reps(\xi',G')$ where $G'=extend(G,U',V')$ in $O(f_0(m,n)+n)$ time.*

Proof. Let $G=(U,V,E)$ belong to C, let R be a representation in $reps(\xi,G)$, let U' be a subset of U, let V' be a subset of V, and let $G'=extend(G,U',V')$. We construct the desired ξ' as follows. For each representation R in $reps(\xi,G)$, the set $reps(\xi',G')$ includes a representation R' that extends R by incorporating a simple adjacency list representation of the dummy vertices and their incident edges.

The elimination ordering associated with R' is obtained by prepending the dummy left (resp., right) vertices to the sorted list of left (resp., right) vertices associated with the elimination ordering of R. The correctness of this approach follows from repeated application of Lemma C.2.

The complexity claims made in the lemma with respect to ξ' are all straightforward to verify. □

D A Unit-Demand Auction

Consider an auction in which many different items are for sale. Assume that a bidding agent assigns a separate value to each item, and is interested in acquiring at most one item. Such an agent is said to have unit-demand preferences. In a unit-demand auction, the bid of an agent takes the same form as a unit-demand preference function: The agent specifies an offer for each item, with the understanding that the bid can win at most one item.

A Vickrey-Clarke-Groves (VCG) mechanism can be used to determine a suitable outcome — allocation and pricing — for any sealed-bid unit-demand auction. (The reader is referred to the survey of Nisan [13] for an introduction to mechanism design, including VCG mechanisms.) The set of possible VCG allocations corresponds to the set of maximum-weight matchings in the bipartite graph that encodes the unit-demand bids: there is a left vertex for each agent; there is a right vertex for each item; if the unit-demand bid of agent u includes an offer for item v, then there is an edge (u,v) with weight equal to u's offer for v. (In the current presentation we make the

standard assumption that the edge weights are all nonnegative; this assumption is not essential, but it simplifies our discussion.)

The VCG mechanism guarantees efficiency, individual rationality, and strategyproofness. Strategyproofness ensures that an agent u is motivated to submit a unit-demand bid equal to the actual preferences of u. In the context of unit-demand auctions, the VCG mechanism also achieves envy-freedom: No agent can achieve a higher utility by receiving a different allocation at the specified prices. It is known that the envy-free price vectors form a complete lattice with respect to the pointwise minimum and maximum operations. Hence there is a unique minimum envy-free price vector; this price vector is known to coincide with the VCG prices. (The reader is referred to Roth and Sotomayor [15, Chapter 8] for proofs of various foundational results related to unit-demand auctions.)

The purpose of this section is to consider the special case of unit-demand auctions where the agent preferences correspond to a vertex-weighted 2DORG: there is a left vertex for each agent; there is a right vertex for each item; each item has two associated values that we refer to as an x-value and a y-value (e.g., third-party "quality" and "seller reliability" scores); the bid of an agent u is encoded as a triple (w, x', y') which means that u is willing to pay w for any item with x-value at least x' and y-value at least y'. Thus the bid graph corresponds to a 2DORG with weights on the left vertices, and Theorem 1 implies that a VCG allocation can be computed in $O(n \log^2 n)$ time. Below we argue that, given such a VCG allocation, the VCG prices can be computed $O(n \log n)$ time.

In order to compute the VCG prices in $O(n \log n)$ time, we exploit the aforementioned characterization of the VCG price vector as the minimum envy-free price vector. Our plan is to start with a price of zero for each item and iteratively increase individual prices — while maintaining the invariant that no item price exceeds the VCG price — until an envy-free price vector is reached. Assuming that the desired invariant can be maintained, we are assured that the final prices are equal to the VCG prices. In order to maintain the invariant, each time we raise the price of an item v from p to p', we establish that the price of item v is at least p' in any envy-free solution. We increase the item prices in two phases, as described below.

In the first phase, we set the tentative price of each item v to the maximum vertex weight of any nonallocated agent u who is bidding on item v (i.e., any agent u such that the x-value of v is at least that of u and the y-value of v is at least that of u). This can be accomplished by processing all of the nonallocated agents in nonincreasing order of their bids (i.e., vertex weights). To process a nonallocated agent u with associated vertex weight w, we repeatedly perform $delete\text{-}min(u)$ operations until nil is returned, setting the tentative price of each item returned by this sequence of operations to w. The total time complexity of this phase is $O(n \log n)$. Since the VCG prices are envy-free, it is easy to see that the tentative price of an item v at the end of the first phase is at most its VCG price.

In the second phase, we further increase the tentative prices as necessary to ensure that the allocated agents are envy-free. We begin by restoring the bid graph to its state prior to phase one. We maintain a priority queue containing all of the allocated agents, where the priority of an agent is given by the current tentative price of its allocated item. We iteratively remove and process a highest-priority allocated agent u from the priority queue. To process such an agent u, we repeatedly perform $delete\text{-}min(u)$ operations until nil is returned, setting the tentative price of each item returned by this sequence of operations to the tentative price of the item allocated to u. It is easy to argue that the sequence of prices associated with all of tentative price updates performed

by the algorithm (across all iterations) is nonincreasing, and that whenever an allocated agent u is removed from the priority queue, the tentative prices of each remaining item (i.e., each item that has yet to be deleted from the bid graph) is at most the tentative price of the item allocated to u. It follows that no tentative price is ever decreased, and that the tentative price of the item allocated to u has attained its final value by the time u is removed from the priority queue. Recall that whenever we increase the tentative price of an item v from p to p', we need to argue that the VCG price of item v is at least p'. This claim is easy to establish by induction on the number of price increases: if in processing an item u with allocated item v we increase the price of an item v' from p to p', it is because the tentative price of v is p', and hence the induction hypothesis implies that the VCG price of v is at least p', and hence envy-freedom implies that the VCG price of v' is at least p'. Since the tentative price of an item is increased at most once, the total time complexity of all operations associated with the priority queue is $O(n \log n)$. Since each $delete\text{-}min(u)$ operation has $O(\log n)$ time complexity, and the number of such operations performed is $O(n)$, the overall time complexity of the second phase is $O(n \log n)$.

To complete our proof, it remains only to argue that upon termination of the second phase, the prices are envy-free. The definition of the first phase ensures that every nonallocated agent is envy-free (i.e., is content with not receiving an item) at the end of the first phase. Since prices do not decrease in the second phase, every nonallocated agent remains envy-free at the end of the second phase. Now consider an arbitrary allocated agent u and assume that item v is allocated to u. Let item v have tentative price p when u is removed from the priority queue. Since the sequence of price updates performed by the algorithm is nonincreasing, any item v' deleted prior to the processing of u has final price at least p. Furthermore, the processing of agent u ensures that any item v' not deleted prior to the processing of u, and that meets the bidding constraints of u (i.e., such that edge (u, v') belongs to the bid graph), has final price exactly p. Thus u is envy-free (i.e., is content with receiving item v) upon termination of the second phase. We conclude that our two-phase algorithm computes the VCG prices.

For the sake of concreteness, we have focused the foregoing discussion on the special case of unit-demand auctions with a 2DORG structure. Our arguments immediately generalize to give the following theorem.

Theorem 3. *Let ξ denote a representation scheme for a hereditary class of bipartite graphs with delete-min complexity $f(n)$. Assume that we are given a representation in $reps(\xi, G)$ of a bipartite graph $G = (U, V, E)$ along with an LMWM of G. Interpreting G as a bid graph for a unit-demand auction, the VCG prices for the items in V can be computed in $O(nf(n) + n \log n)$ time.*

In auction settings it is often desirable to allow a seller to specify a reserve price, that is, a lower bound on the selling price. For unit-demand auctions, reserve prices can be modeled by introducing a suitable dummy bidder for each item. Using Lemma C.3 in conjunction with Theorem 3, it is easy to argue that reserve prices can be accommodated in our setting without affecting the asymptotic time complexity of the auction.