Dominance Drawings of Bipartite Graphs*

Peter Eades(1) Hossam ElGindy(1) Michael Houle(1) Bill Lenhart(2)
Mirka Miller(1) David Rappaport(3) Sue Whitesides(4)

(1)Department of Computer Science

The University of Newcastle, New South Wales, Australia.

(2)Department of Computer Science

Williams College, Williamstown, USA.

(3)Department of Computing and Information Science

Queen's University, Kingston, Ontario, K7L 3N6, CANADA.

(4)School of Computer Science

McGill University Montreal, Quebec, CANADA.

April 29, 1994

Abstract

Let $G = (S, T, E)$ denote a directed bipartite graph with S a set of sources and T a set of sinks. Using vectors $\mathbf{s} = (s_1, s_2, ...s_k)$ and $\mathbf{t} = (t_1, t_2, ...t_k)$ in R^k to represent the elements $s \in S$ and $t \in T$ we can geometrically characterize the properties of G. We consider the case where $s_i \leq t_i$, $i = 1, 2, ..., k$, (with strict inequality in at least one coordinate) if and only if the directed edge $(s, t) \in E$. Such an assignment of coordinates to vertices is known as a *dominance drawing* of G, because all edges in the graph are geometrically characterized by the dominance relation. We present an asymptotically optimal algorithm to obtain a 2-dimensional dominance drawing of a bipartite graph whenever such a drawing exists.

*Part of the work was carried out when the authors were participants of the Workshop on Layout and Optimal Path Problems at Hawks Nest, Australia, sponsored by the Department of Computer Science, The University of Newcastle, New South Wales, Australia

1 Introduction

Geometric representations can be valuable tools for characterizing properties of graphs. A good drawing of a graph is a useful representation that aids in our understanding of interdependencies of components for a variety of applications. This can include the design of networks, or programs, assigning or scheduling tasks, or navigating through a database. See [6] for a comprehensive survey of methods that can be used to obtain good graph drawings. We will consider graphs that represent partial orders and discuss a particular method of drawing the graphs to geometrically display the transitive closure property of the partial order.

We begin with some definitions. Let $G = (V, E)$ denote a directed graph with vertex set V and edge set $E = \{(x, y) : x, y \in V\}$. We say that the edge (x, y) is *directed* from x to y. A vertex y is *reachable* from x if there is a *path* of one or more directed edges that lead from x to y. An unreachable vertex is a *source* and an unreaching vertex a *sink*. A *directed acyclic digraph* (DAG) is a directed graph with no vertex reachable from itself. A directed edge (x, y) is *transitive* if y is also reachable from x by a path not using (x, y). A *transitive digraph* is a DAG that includes all transitive edges. Consider a transitive DAG G. We can use vectors $\mathbf{x} = (x_1, x_2, ...x_k)$ to represent each vertex x of G, so that $x_i \leq y_i, i = 1, 2, ..., k$, (with strict inequality in at least one coordinate) if and only if y is reachable from x in G. We call such an assignment of coordinates to vertices a *dominance drawing*, because all edges in the graph (and transitive closure) are geometrically characterized by the dominance relation. The value of the *dominance drawing dimension* of G, $d(G)$, is the value of the smallest k for which a k-dimensional dominance drawing of G can be obtained.

In this paper we present results pertaining to transitive directed bipartite graphs. Let $G = (S, T, E)$ denote a directed bipartite graph with S a set of *sources* and T a set of *sinks*, so that all edges (s, t) are directed from a source to a sink. The transitive property of G is trivially realized since the maximum path length in G is one. Let $n = min(|S|, |T|)$. In section 2 we show that $d(G) \leq n$. In section 3 we show that a two dimensional dominance drawing for a transitive bipartite graph can be obtained in polynomial time. The complexity of our algorithm is linear in the size of the input and is thus optimal. We conclude the paper in section 4 by discussing the history of some related results and by posing some open problems.

2 Upper and lower bounds on the dominance drawing dimension of transitive bipartite graphs

Let $G = (S, T, E)$ be a transitive bipartite graph with $S = (s_1, s_2, \ldots, s_{|S|})$ a set of sources and $T = (t_1, t_2, \ldots, t_{|T|})$ a set of sinks. Let $n = min(|S|, |T|)$. We will show that $d(G) \leq n$, by assigning

n-dimensional vectors to the vertices of G that satisfy the dominance requirements. The assignment differs depending on whether $|S| \leq |T|$ or not. We start with an assignment that uses $|S|$ dimensions. The key idea is to use two dimensions to obtain a dominance drawing of a subgraph of G induced by two sources and all of the sinks. The remaining $|S|-2$ dimensions are used for the rest of the graph. We will use the notation s_j to denote the jth coordinate of element s. Let $N(s) = \{t : (s, t_j) \in E\}$ be used to denote the *neighbourhood* of s. We say that the neighbourhood of s is *empty* if $|N(s)| = 0$

Lemma 2.1 $d(G) \leq |S|$.

Proof: We begin by considering two sources s_1 and s_2 with nonempty neighbourhoods. If there is only one source with a non-empty neighbourhood then the lemma has a trivial proof. We can label the vertices in $|T|$ with integers from $1 \dots |T|$, so that we have integers $1 \leq \alpha \leq \beta \leq \gamma \leq |T|+1$, and $(s_1, t_j) \in E$ if and only if $1 \leq j < \beta$, and $(s_2, t_j) \in E$ if and only if $\alpha \leq j < \gamma$.

The vector assignments for T and S are:

$$t_{j,i} = \begin{cases} j, & i = 1 \\ |T| - j, & i = 2 \\ 1, & i \geq 3, (s_i, t_j) \in E \\ 0, & i \geq 3, (s_i, t_j) \notin E \end{cases} \qquad s_{i,j} = \begin{cases} 0, & j = 1, i = 1 \\ \alpha - 1/2, & j = 1, i = 2 \\ i - |S|, & j = 1, i \geq 3 \\ |T| - \beta + 1, & j = 2, i = 1 \\ |T| - \gamma + 1/2, & j = 2, i = 2 \\ -i, & j = 2, i \geq 3 \\ 1, & j \geq 3, i = j \\ 0, & j \geq 3, i \neq j \end{cases}$$

We now verify that this assignment satisfies all requirements. Observe that $t_{1,i}$ increases with i and $t_{2,i}$ decreases with i so no elements in T dominate each other. The similar situation is found among elements of $\{s_1, s_2\}$ and the set $S - \{s_1, s_2\}$. The elements s_1 and s_2 dominate $S - \{s_1, s_2\}$ in the first two coordinates, but, for $3 \leq i \leq |S|$, s_i dominates both s_1 and s_2 in the ith coordinate, so there is no dominance among elements of S.

It remains to show that t_j dominates s_i in all coordinates if and only if $(s_i, t_j) \in E$. For s_1 and s_2, we use the first two dimensions to determine dominance. We have ordered T and chosen locations for s_1 and s_2 accordingly. For the rest of the set S, every element of T is set to dominate strictly in the first two coordinates, and to dominate with possible equality in all coordinates except the ith. It is now easy to see that a particular t_j will dominate s_i in the ith coordinate if and only if $(s_i, t_j) \in E$. □

We can accomplish the same effect by isolating two sinks, assigning vectors to satisfy dominance among the two sinks and all of the sources in two dimensions, and then use $|T| - 2$ dimensions to satisfy the remaining requirements. Thus we can conclude.

Theorem 2.2 *Given a transitive bipartite graph $G = (S, T, E)$, $d(G) \leq min(|S|, |T|)$.*

The proof found in [8] can be used to obtain our result. That is:

Theorem 2.3 (Hiraguchi) *Given a transitive acyclic digraph, $G = (V, E)$, $d(G) \leq |V|/2$.*

We have included our proof for the bipartite case because of its relative simplicity, and because it provides good intuition for the algorithm presented in the next section.

We should mention that these bounds are tight for the general case as well as the bipartite case as proved by [8] and [15]. An example of a bipartite graph G with $2n$ vertices such that $d(G) = n$, as proved in [15], is: $G = (S, T, E)$ with bipartition $S = (s_1, s_2, \ldots, s_n)$, and $T = (t_1, t_2, \ldots, t_n)$, and edges (s_i, t_j) for all i and j, $i \neq j$.

3 Two Dimensional Dominance Drawings of Transitive Bipartite Graphs

We present an algorithm that obtains a two dimensional dominance drawing of a transitive bipartite graph whenever such a drawing exists. The running time of the algorithm is proportional to the size of the input and is thus optimal. Let $G = (S, T, E)$ be a transitive bipartite graph with $S = (s_1, s_2, \ldots, s_{|S|})$ a set of sources and $T = (t_1, t_2, \ldots, t_{|T|})$ a set of sinks. Consider the collection $\mathcal{N} = \{N(s) : s \in S\}$, of subsets of T, let $\mathcal{M} = \{N(s) - N(w) : s, w \in S, N(w) \subset N(s)\}$, and let $\mathcal{I}(S,T) = \mathcal{N} \cup \mathcal{M}$. Finally, let $\Pi(\mathcal{I}(S,T))$ denote the collection of all permutations π of T such that members of each set $I \in \mathcal{I}(S,T)$ are contiguous in π. Our algorithm to obtain a two dimensional dominance drawing of the graph G is based on the following characterization of two dimensional transitive bipartite graphs.

Theorem 3.1 *The dimension of $G = (S, T, E)$ is less than or equal to two if and only if $\Pi(\mathcal{I}(S,T))$ is not empty.*

Before proving the theorem we provide some preliminary definitions. We use $\pi(i)$ to denote the sink at position i in π, and $\pi^{-1}(t)$ to denote the position in π of the sink labeled t. We use the notation $left(s)$ and $right(s)$ to respectively denote the position of the left and right most elements of $N(s)$ in π. We begin by showing that every two dimensional dominance drawing of G can be represented by a permutation that satisfies the requirements of Theorem 3.1.

Lemma 3.2 *If $G = (S, T, E)$ is a bipartite graph such that $\Pi(\mathcal{I}(S,T))$ is not empty, then $d(G) \leq 2$.*

Proof: Let π be any permutation in $\Pi(\mathcal{I}(S,T))$. We place sinks in the order in which they appear in π on the line $y = |T| - x$, that is, $t_{j,1} = \pi^{-1}(t_j)$, and $t_{j,2} = |T| - \pi^{-1}(t_j)$. For each

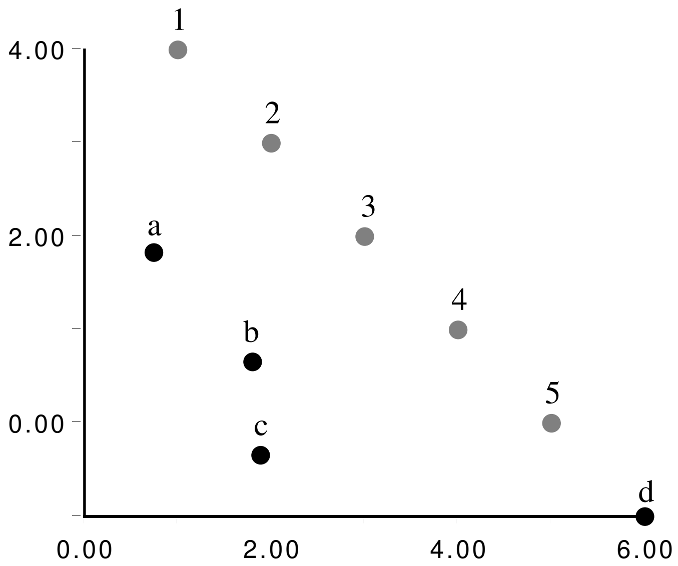

Figure 1: A two dimensional dominance drawing of the graph $S = \{a, b, c, d\}$, $T = \{1, 2, 3, 4, 5\}$, and $N(a) = \{1, 2, 3\}$, $N(b) = \{2, 3, 4\}$, $N(c) = \{2, 3, 4, 5\}$, $N(d) = \emptyset$.

source s_i with a non-empty neighbourhood we assign the coordinates $s_{i,1} = left(s_i) - \frac{1}{right(s_i)+1}$ and $s_{i,2} = |T| - right(s_i) - \frac{left(s_i)}{|T|+1}$. For sources with an empty neighbourhood we can use any point that neither dominates nor is dominated by any other. The point $(|T| + 1, -1)$ is a suitable choice. See figure 1 for an example.

To verify that we have a valid dominance drawing observe that since all sinks are on the line $y = |T| - x$, no two sinks dominate each other. The only sinks in the drawing that dominate a source s are those sinks in the substring of π, $(\pi(left(s)) \ldots \pi(right(s)))$ because sink t dominates s, if and only if $left(s) \leq \pi^{-1}(t) \leq right(s)$. Finally to show that no two sources dominate each other consider two sources s and w. If w dominates s then $w_1 \geq s_1$ and $w_2 \geq s_2$. This can only occur if either

1. $left(w) > left(s)$ and $right(w) < right(s)$, or

2. $left(w) = left(s)$ and $right(w) = right(s)$.

However, the first possibility implies that $N(w) \subset N(s)$ and that $N(s) - N(w)$ is not contiguous, and this cannot occur. The second possibility implies that $N(w) = N(s)$, in which case s and w are drawn in the same location and so neither dominates the other. Thus we have verified that these assignments result in a two-dimensional dominance drawing. $\square$

Lemma 3.3 *If $d(G) \leq 2$ then $\Pi(\mathcal{I}(S, T))$ is not empty.*

Proof: Consider any two dimensional dominance drawing of G. Observe that if we sort the sinks from left to right we get the same unique sequence as sorting from top to bottom, because no sink dominates any other. For simplicity we assume that t_j is the jth element in a left to right sorted sequence of the sinks (and the jth element sorted from top to bottom). We claim that $\pi = (t_1, t_2, \ldots, t_{|T|})$ is a permutation in $\Pi(\mathcal{I}(S,T))$. We first verify that $N(s)$ is contiguous in π for all $s \in S$, by showing that every element between the positions $left(s)$ and $right(s)$ in π dominates s in the drawing. Consider any element $t_k \in T$ in position k in π where $left(s) < k < right(s)$. Thus $t_{k,1} > t_{left(s),1} \geq s_1$ and $t_{k,2} > t_{right(s),2} \geq s_2$ and therefore dominates s. So all sinks t that dominate s in the dominance drawing are contiguous in π, thus $N(s)$ is contiguous in π. Consider a $w \in S$ such that $N(w) \subset N(s)$. We must show that $N(s) - N(w)$ is contiguous in π. Suppose for the sake of contradiction that $N(s) - N(w)$ is not contiguous in π. This implies that $left(s) < left(w)$, and $right(s) > right(w)$. Observe that $t_{left(s),2} > t_{left(w),2} \geq w_2$. Therefore, since $t_{left(s)}$ dominates s but does not dominate w we must have $s_1 \leq t_{left(s),1} < w_1$. By a similar argument we can show that $s_2 \leq t_{right(s),2} < w_2$, and this implies that w dominates s, which leads us to the desired contradiction. Thus we have shown that every two-dimensional dominance drawing of $G = (S, T, E)$ can be represented by a permutation $\pi \in \Pi(\mathcal{I}(S,T))$ $\square$

The proof of Theorem 3.1 follows as a direct consequence of Lemma 3.2 and Lemma 3.3.

Consider a set X and a set Ξ of subsets of X. Booth and Leuker [3] present a data structure called a *PQ-tree* that can be used to determine the set of all permutations π of X having the property that every subset $\xi \in \Xi$ is contiguous in π. PQ-trees allow this set of permutations to be found in time linear in the size of the input. Thus it appears that an expedient solution to our problem is to compute the set $\mathcal{M}$ and subsequently the set $\mathcal{I}(S,T)$ and apply the PQ-tree algorithm. However, the size of $\mathcal{M}$ may be $O(|S|^2)$, too big for a final complexity that is proportional to the size of the original input. This problem is not insurmountable as we can restrict our attention to an $O(|S|)$ subset of $\mathcal{M}$. Consider the case where we have a maximal sequence of elements of S such that, $N(s_1) \subset N(s_2) \subset \cdots \subset N(s_k)$. Then we only need to consider the set $N(s_k) - N(s_1)$. However, the task of computing this reduced subset of $\mathcal{M}$ in linear time, that is, without explicitly sorting, approaches the conceptual difficulty of presenting our new algorithm. Therefore, we present a new algorithm and skirt the problem of computing a reduced subset of $\mathcal{M}$. The data structure we use to represent the family of permutations $\Pi(\mathcal{I}(S,T))$ is influenced by the PQ-tree, but it is much simpler.

We define a *box list* of a set T, $\mathcal{B}(T)$, as the empty list, or a linked list consisting of one or more subsets of T, called *boxes*, such that the boxes form an exact cover of T; that is, the union of the boxes in the box list equals T and each element of T appears in exactly one box. If we give an ordering for the elements of each box, then a rear to front (or front to rear) traversal of $\mathcal{B}(T)$ corresponds to a permutation of T. A permutation π of T is *consistent* with $\mathcal{B}(T)$ if the elements

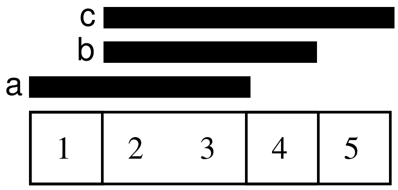

Figure 2: The graph $G = (S, T, E)$, $d(G) = 2$. $N(a) = \{1, 2, 3\}$, $N(b) = \{2, 3, 4\}$, and $N(c) = \{2, 3, 4, 5\}$. We illustrate a permutation in $\Pi(\mathcal{I}(S, T))$ using intervals. We also show the box list representation of this permutation.

within boxes can be ordered so that a traversal of $\mathcal{B}(T)$ is equal to π. Let $F(\mathcal{B}(T))$ be used to denote the set of all permutations that are consistent with $\mathcal{B}(T)$ Given a graph $G = (S, T, E)$ our algorithm begins with a box list representing all permutations of T, that is, the box list consists of exactly one box that contains T. If the graph admits a two dimensional dominance drawing then the algorithm exits with a non-empty box list such that $F(\mathcal{B}(T)) \subseteq \Pi(\mathcal{I}(S, T))$, otherwise the algorithm exits with $\mathcal{B}(T) = \emptyset$, the empty list. See figure 2 for an illustrative example.

We assume that G is connected and that no neighbourhood of $s \in S$ is empty and all neighbourhoods are distinct. With these assumptions we can guarantee that our algorithm terminates with $F(\mathcal{B}(T)) = \Pi(\mathcal{I}(S, T))$. There is no loss in generality in making these assumptions because we can process the connected components of G as separate entities and easily merge the solutions. Any elements of S with an empty neighbourhood can be ignored, and all elements of S with the same neighbourhood can ultimately be assigned the same point in the final drawing of G. Elements of S with the same neighbourhoods are easily recognized by our algorithm with no increase in the computational complexity.

The principle operation performed on a box list is to add *constraints* to $\mathcal{B}(T)$ that are associated with a neighbourhood of a source, $N(s)$. The constraints are substrings, or *intervals* in permutations $\Pi(\mathcal{I}(S, T))$. Thus $\mathcal{B}(T)$ is constrained so that the interval associated with $N(s)$ will be contiguous in all permutations of $F(\mathcal{B}(T))$. A high level description of our algorithm can be expressed as follows.

procedure DomDraw$(S, T, E, \mathcal{B}(T))$
$S^* \leftarrow \emptyset$
for all $s \in S$
  $S^* \leftarrow S^* \cup \{s\}$
  Add the constraints associated with $N(s)$ to $\mathcal{B}(T)$ such that $F(\mathcal{B}(T)) \subseteq \Pi(\mathcal{I}(S^*, T))$
  if ($\mathcal{B}(T) = \emptyset$) **then** exit DomDraw
end for
end DomDraw

The subgraph of G induced by $S^* \cup T$ may not be connected so we do not write $F(\mathcal{B}(T)) = \Pi(\mathcal{I}(S^*, T))$. When adding a new constraint $N(s)$ there are two possible outcomes. We add $N(s)$ successfully to $\mathcal{B}(T)$ keeping it and all other members of $\mathcal{I}(S^*, T)$ contiguous, and we claim that $\emptyset \neq F(\mathcal{B}(T)) \subseteq \Pi(\mathcal{I}(S^*, T))$. Alternately, we may find that $N(s)$ cannot be added to $\mathcal{B}(T)$ and then claim that $F(\mathcal{B}(T)) = \Pi(\mathcal{I}(S^*, T)) = \emptyset$.

The key to our algorithm is that we add constraints in a particular order. This ordering reduces the work required. We create this order without doing any explicit sorting. Details regarding the implicit sorting method follow in the complexity analysis of the algorithm. Suppose $d(G) \leq 2$ and $\pi \in \Pi(\mathcal{I}(S, T))$. We define a order relation ρ_π on all pairs $a, b \in S$ as follows: $a \; \rho_\pi \; b$ if either $left(a) < left(b)$, or both $left(a) = left(b)$ and $right(a) < right(b)$.

Lemma 3.4 *Let $a, b, c \in S$ such that $N(a) \cap N(b) \neq \emptyset$, and $N(b) \cap N(c) \neq \emptyset$, and $N(a) \cap N(c) = \emptyset$. For every $\pi \in \Pi(\mathcal{I}(S, T))$, $a \; \rho_\pi \; b$ implies $b \; \rho_\pi \; c$.*

Proof: If $N(a) \cap N(b) \neq \emptyset$, and $N(b) \cap N(c) \neq \emptyset$, and $N(a) \cap N(c) = \emptyset$ and $a \; \rho_\pi \; b$ then $left(a) \leq left(b) < left(c)$ in π which implies that $a \; \rho_\pi \; b \; \rho_\pi \; c$. □

Lemma 3.5 *Let $a, b, c \in S$ such that $N(a) \cap N(b) \cap N(c) \neq \emptyset$. For all $\pi \in \Pi(\mathcal{I}(S, T))$ such that $a \; \rho_\pi \; b$, the relative order of c in ρ_π with respect to a and b is the same.*

Proof: Let us assume that $a \; \rho_\alpha \; b \; \rho_\alpha \; c$. Since $\alpha \in \Pi(\mathcal{I}(S, T))$ and all neighbourhoods of $s \in S$ are assumed distinct, we get $left(a) \leq left(b) \leq left(c)$ and $right(a) \leq right(b) \leq right(c)$, with at least two of the inequalities strict. This implies that either:

$$N(a) \cap N(c) \subset N(a) \cap N(b)$$

or if $left(b) = left(c)$ then

$$N(a) \cap N(c) = N(a) \cap N(b) \text{ and } N(b) \subset N(c).$$

The following are implied as well:

$$N(a) \cap N(c) \subset N(b) \cap N(c)$$

or if $right(a) = right(b)$ then

$$N(a) \cap N(c) = N(b) \cap N(c) \text{ and } N(b) \subset N(a).$$

Now we assume there is some permutation β such that $a \; \rho_\beta \; b$ with a different relative ordering for c. If $a \; \rho_\beta \; c \; \rho_\beta \; b$ then the following inequalities are implied.

$$N(a) \cap N(b) \subset N(a) \cap N(c) \text{ or } N(a) \cap N(b) = N(a) \cap N(c) \text{ and } N(c) \subset N(b)$$

and

$$N(a) \cap N(b) \subset N(c) \cap N(b) \text{ or } N(a) \cap N(b) = N(c) \cap N(b) \text{ and } N(c) \subset N(a).$$

And if there exists a permutation γ such that $c \ \rho_\gamma \ a \ \rho_\gamma \ b$ then the following inequalities are implied.

$$N(c) \cap N(b) \subset N(c) \cap N(a) \text{ or } N(c) \cap N(b) = N(c) \cap N(a) \text{ and } N(a) \subset N(b)$$

and

$$N(c) \cap N(b) \subset N(a) \cap N(b) \text{ or } N(c) \cap N(b) = N(a) \cap N(b) \text{ and } N(a) \subset N(c).$$

These families of inequalities are mutually contradictory, so we conclude that for every permutation π such that $a \ \rho_\pi \ b$ the relative order of c with respect to a and b must be the same.

□

We can determine the relative ordering of the intervals by examining the way in which intervals overlap. If the intervals arise from neighbourhoods of a connected graph, then the relative order of intervals in every permutation of $\Pi(\mathcal{I}(S,T))$ is the same. This makes it possible to efficiently add intervals to the box list, and detect whether the Add is successful or not.

We define a precedence independent of any permutation and based on the overlap of neighborhoods. Let x, y, z be three sets so that $x \cap y \neq \emptyset$ and $x \cap z \neq \emptyset$. Let the precedence $y \prec_x z$ if $|x \cap y| > |x \cap z|$, or if $|x \cap y| = |x \cap z|$ and $|y - (x \cap y)| \leq |z - (x \cap z)|$. This precedence will be used to schedule the Adds.

We are now sufficiently prepared to give a detailed description of the procedure DomDraw. DomDraw calls the procedures Add, Initialize, Sched, and Flip. We also use a procedure Enqueue (and subsequently Dequeue) which is a standard insert into the front (deletion from the rear) of a queue.

procedure DomDraw(S,T,E,B(T))
{We assume that the input represents a connected transitive bipartite DAG. }
$\mathcal{B}(T) \leftarrow$ one box consisting of the entire set T
$Q \leftarrow \emptyset$ {Set up an empty queue Q}
Choose an $s_{start} \in S$ so that $|N(s_{start})|$ is maximized
$S^* \leftarrow \{s_{start}\}$; Add(s_{start}) ; Enqueue(s_{start}, Q)
Initialize($\mathcal{B}(T), S^*, H, Q$)
{ H is a subset of S that we want to ignore for the first round of Sched.}
{However, we'll need to look at the elements of H later.}
$S \leftarrow S - H$
Sched($\mathcal{B}(T), S^*, Q$)
{At this point we have built the box list from left to right starting at s_{start}. }
{Flip things around so that we can go the other way. We put the elements of H. }
{back into S and resume the Sched procedure with s_{start} in the queue. }
Flip($\mathcal{B}(T)$) ; $S \leftarrow S + H$; Enqueue(s_{start}, Q)

Sched($\mathcal{B}(T), S^*, Q$)
end DomDraw

The schedulers Initialize and Sched are specified as follows.

procedure Initialize($\mathcal{B}(T), S^*, H, Q$)
$s \leftarrow$ Dequeue(Q)
$W'(s) \leftarrow \{w' \in S - S^* : N(w') \cap N(s) \neq \emptyset\}$
Find a maximal subsequence of $W'(s) \supseteq W(s) = (w_1, w_2, \ldots, w_k)$, such that $(N(s) \cap N(w_i)) \supseteq (N(s) \cap N(w_j))$, if $i < j$
$H = W'(s) - W(s)$
Arrange the sequence $(w_1, w_2, \ldots, w_k)$ so that $N(w_i) \prec_{N(s)} N(w_j)$, if $i < j$
for $i \leftarrow 1 \ldots k$
    $S^* \leftarrow S^* \cup \{w_i\}$
    Add($N(w_i)$)
    Enqueue(w_i, Q)
    end for
end Initialize

Once the box list is initialized we continue to add constraints. Intervals are always appended to a well structured box list in the procedure Sched.

procedure Sched($\mathcal{B}(T), S^*, Q$)
while ($Q \neq \emptyset$) **do**
    $s \leftarrow$ Dequeue(Q)
    $W(s) \leftarrow \{w \in S - S^* : N(w) \cap N(s) \neq \emptyset\}$
    Arrange $W(s)$ to obtain a sequence $(w_1, w_2, \ldots, w_{|W(s)|})$ such that $N(w_i) \prec_s N(w_j)$, if $i < j$
    for $i \leftarrow 1 \ldots |W(s)|$
        $S^* \leftarrow S^* \cup \{w_i\}$
        Add($N(w_i)$)
        Enqueue(w_i, Q)
    end for
end while
end Sched

After one successful run of procedure Sched there may be some elements of S that have not been processed because they don't intersect with any interval represented in S^*. We use the procedure Flip to facilitate processing the remaining intervals.

procedure Flip($\mathcal{B}(T)$)
Let $b_1, b_2, \ldots, b_k, lastbox$ represent a left to right traversal of the boxes in the box list $\mathcal{B}(T)$
Reorganize $\mathcal{B}(T)$ to get $b_k, b_{k-1}, \ldots, b_1, lastbox$
end Flip

In procedure Add we use *lastbox* to identify the box in $\mathcal{B}(T)$ with no successors, and we use $fb(x)$ to denote the first box from the left in $\mathcal{B}(T)$ that contains an element of the set x.

```
procedure Add(x)
{Add the constraints, associated with the set x ⊆ T, to the box list. }
b←fb(x); z←x
if ( z ∩ b ≠ b ) then
   Create an empty box a so that it is the successor of b in B(T)
   a←z ∩ b; b←b − a
end if
z←(z − b); b←succ(b)
 { Scan the box list from left to right until the first non element of x is encountered. }
while (  b ∩ z = b and b ≠ lastbox  ) do
   z←(z − b); b←succ(b)
end while
if ( b = lastbox ) then
   if ( z ≠ ∅ and z ≠ b ) then
      Create an empty box a so that it precedes b in B(T)
      a←z; b←b − a
   end if
else
      B(T)←∅ ; exit DomDraw
end if
end Add
```

The procedure DomDraw adds intervals to the box list optimistically assuming that the dimension of the input graph $d(G) \leq 2$. In the event that this is not the case we must ensure that unsuccessful Adds are correctly determined. The order in which the Adds are performed guarantee this property.

Lemma 3.6 *Assume $N(s) \cap N(u) \cap N(w) \neq \emptyset$, $N(u) \prec_{N(s)} N(w)$ and $N(s) \not\subseteq (N(u) \cup N(w))$. There are permutations $\pi \in \Pi(\mathcal{I}(S,T))$ such that $s\ \rho_\pi\ u$ implies either $s\ \rho_\pi\ u\ \rho_\pi\ w$ or $d(G) > 2$.*

Proof: The interval corresponding to $N(w)$ can fit in one of three places in any of these permutations π. We enumerate these possibilities.

Suppose that $s\ \rho_\pi\ w\ \rho_\pi\ u$, then by Lemma 3.5 we have $N(s) \cap N(w) \subseteq N(s) \cap N(u)$. This contradicts the assumption that $N(u) \prec_{N(s)} N(w)$. On the other hand assume that $w\ \rho_\pi\ s\ \rho_\pi\ u$, then by Lemma 3.5 $N(w) \cap N(u) \subseteq N(w) \cap N(s)$. This contradicts the assumption that $N(s) \not\subseteq (N(u) \cup N(w))$ and $N(u) \prec_{N(s)} N(w)$. Therefore, the only place where w fits is $s\ \rho_\pi\ u\ \rho_\pi\ w$.

□

We maintain the following conditions prior to every invocation of procedure Add.

Lemma 3.7 *Let $s \in S^*$ and $w \in W(s)$. Prior to adding the interval $N(w)$ to $\mathcal{B}(T)$ we ensure that if there exists $u \in S^*$ such that $u \in W(s)$ and $N(u) \prec_{N(s)} N(w)$, then $N(s) \not\subseteq (N(u) \cup N(w))$.*

Proof: The condition holds when we add to s_{start} in Initialize, because $|N(s_{start})|$ is maximal and the reduced set of neighbours do not contain s_{start} in their union. When we add to s_start after the *Flip* we add the elements of the set H, and again we are guaranteed that the union of these elements do not contain s_{start}.

Consider all other Adds invoked from procedure Sched. Let $s \in S^*$ and $u, w \in W(s)$, such that $u \prec_{N(s)} w$. In this case there exists an s' such that $N(s') \cap N(s) \neq \emptyset$ and $N(s') \cap N(u) = \emptyset$ and $N(s') \cap N(w) = \emptyset$. Therefore, $N(s) \not\subseteq (N(u) \cup N(w))$. □

Lemma 3.8 *Add returns $F(\mathcal{B}(T)) \subseteq \Pi(\mathcal{I}(S^*, T))$.*

Proof: The first call to Add, in procedure DomDraw, adds an interval to an empty box list. Thus, we initially have $F(\mathcal{B}(T)) \subseteq \Pi(\mathcal{I}(S^*, T))$ for the single interval $s_{start} \in S^*$ As an inductive hypothesis assume that prior to adding $N(w)$ we have $F(\mathcal{B}(T)) \subseteq \Pi(\mathcal{I}(S^* - \{w\}, T))$.

We examine, in detail, the way in which an interval $x = N(w)$ is handled. The Add procedure scans the box list from left to right to ensure that the elements of x are contiguous. The first box that overlaps with x may not be contained in x, in which case we reduce the number of permutations in $F(\mathcal{B}(T))$ by splitting the box into a part which contains no elements of x and a part which is contained in x. The scan then continues using a while loop until the first non element of x is encountered in $\mathcal{B}(T)$ or the lastbox in $\mathcal{B}(T)$ is encountered.

If the scan ends at the lastbox then the constrained part of the box list, that is, all elements of x in the boxes to the left of the lastbox, are contiguous within the interval x. The part of x, if any, that overlaps the lastbox, determines the way in which the lastbox is split. Clearly, the interval x is contiguous in every permutation of $F(\mathcal{B}(T))$. Our scheduling of the adds guarantees that any interval y that contains x has already been added to $\mathcal{B}(T)$ and since the first non element of x encountered in the scan is in the lastbox of $\mathcal{B}(T)$, then for all y that contain x, $y - x$ is contiguous. Therefore, $\Pi(\mathcal{I}(S, T)) = \emptyset$.

Suppose, on the other hand, that in our scan of $\mathcal{B}(T)$ the first non element of x is found in a box b that precedes lastbox. We know that the conditions of 3.7 are maintained and thus we can use Lemma 3.6 to reason about unsuccessful Adds. If there are any elements of x that have not been seen in the scan of $\mathcal{B}(T)$ then we conclude that there is no permutation in $F(\mathcal{B}(T))$ in which x is contiguous. Consider now the case where the scan has seen all of the elements of x. Let y be the interval that was added to $\mathcal{B}(T)$ immediately before x, then we claim that $y - x$ is not contiguous in any permutation of $F(\mathcal{B}(T))$. The justification of this claim is a result of the scheduling of the adds which implies that $left(y) < left(x)$ and $right(y) > right(x)$, in every permutation of $F(\mathcal{B}(T))$.

Therefore after we Add $N(w)$ we have $F(\mathcal{B}(T)) \subseteq \Pi(\mathcal{I}(S^*, T))$.

$\square$

Recall that we have assumed that the input consists of a connected graph. Therefore, at the end of the successful conclusion of algorithm DomDraw the box list represents all of the permutations in $\Pi(\mathcal{I}(S, T))$. Our algorithm development is summarized in the following theorem.

Theorem 3.9 *Given a connected transitive bipartite graph $G = (S, T, E)$, procedure DomDraw exits with $F(\mathcal{B}(T)) = \Pi(\mathcal{I}(S, T))$.*

We now consider the implementation and computational complexity analysis of algorithm DomDraw.

Lemma 3.10 *Algorithm DomDraw can be implemented to run in $O(|S|+|T|+|E|)$ time and space, and this is within a constant multiple of optimal.*

Proof: The main computational steps involve reading the input, the computation of the sets $W(s)$, sorting the sets $W(s)$, and performing Adds.

Let $G' = (S', T', E')$ be used to denote a copy of the graph G with undirected edges. A traversal of G' is used to compute $W(s)$. For every $t \in N(s)$ we visit every vertex w, such that $(w, t) \in E'$. We then add w to $W(s)$ and delete the edge (w, t) from E'. In this way we can obtain $W(s)$ and keep track of those elements in each $N(w)$ that overlap $N(s)$, that is, $N(s) \cap N(w)$. Thus the expense of computing $W(s)$ is proportional to the sum of the edges emanating from the vertices in $W(s)$. Each such edge is used once and then deleted so this work amounts to $O(|E|)$ throughout the life of the algorithm.

To sort the elements of $W(s)$ we determine the way in which they overlap. Consider the sinks t in the union of the neighbourhoods of $W(s)$. Let $c_1(t) = |\{w \in W(s) : t \in N(w) \text{ and } t \in N(s)\}|$. Similarly we define a second cost $c_2(t) = |\{w \in W(s) : t \in N(w) \text{ and } t \notin N(s)\}|$. Let $R_1(w) = \min_{t \in N(w)} c_1(t)$ and let $R_2(w) = \min_{t \in N(w)} c_2(t)$. The cost to compute c_1, c_2, R_1, and R_2 is proportional to the size of the subgraph induced by the vertices in $W(s)$ and their neighbourhoods. This amounts to $O(|E|)$ throughout the life of the algorithm. The values of $R_1(w)$ and $R_2(w)$ are in the range $1 \ldots |W(s)|$. Thus we can use a table of length $|W(s)|$ to rank these values in $O(|W(s)|)$ time and space complexity. Let L_1 be a table of length $|W(S)|$, where each element of L_1 is a linked list. The elements of $W(s)$ with the rank $R_1(w) = k$ are put in the list at the entry k of L_1. We create a similar table L_2, where the $R_2(w)$ rankings are used to assign locations in L_2. A linear scan of L_1 produces the sorted $W(s)$ if there are no ties among the R_1 values, that is, every entry in L_1 is either an empty list or a single value. If ties need to be broken we use the values in the table L_2. We again can bound the cost of all such table scans by $O(|E|)$.

Regarding the complexity of performing an Add into $\mathcal{B}(T)$, we keep a dictionary indexed by integers from $1 \ldots |T|$ for locating elements t, in the box list. From the dictionary we can obtain, in constant time, the pointer to the element t in T, as well as a reference to the box that contains t. We do $O(|N(w)|)$ of these lookups for an Add$(N(w))$. For each box b that contains a $t \in N(w)$ we create an empty list h. Whenever we locate a t from b add it to the list h. Thus each h represents the set $z \cap b$ as described in procedure Add. If $h = b$ then we conclude that $z \cap b = b$. If $z \cap b \neq b$ then we have either discovered that $d(G) > 2$, or for the case where $b = fb(x)$ or $b = lastbox$ we split a box into two sub-boxes. In the box splitting situation we assign $a \leftarrow h$ and $b \leftarrow b - h$. This can be done with pointer operations in constant time. Thus we conclude that the complexity of procedure Add is $O(|N(w)|)$, and this results in a total complexity of $O(|E|)$ for procedure DomDraw.

The size of the input of the graph is $O(|S| + |T| + |E|)$, and we must perform on the order of this amount of work to read it. Thus the total cost of our algorithm is $O(|S| + |T| + |E|)$ and since the complexity is dominated by reading the input, our algorithm is optimal within a constant factor. □

4 Discussion

We have given a linear time algorithm to obtain a 2D dominance drawing whenever such a drawing exists. No complicated data structures are used, so the algorithm should be easy to implement. It should be pointed out that a very similar approach was used to recognize the so called bipartite permutation graphs in linear time [13]. It turns out that bipartite permutation graphs and bipartite graphs that admit a two dimensional dominance drawing have identical characterizations. So the results of this paper are implicit from the results of [13]. In fact there is a rich history relating to this problem, and we review it here briefly.

A partial order P of a finite set X is a transitive and non-reflexive binary relation on X. A partial order can be represented by a *transitive digraph*, $G(P)$, on the elements of X, The *dimension* $d(P)$ of a partial order P [5] is the minimum number of linear orders whose intersection is P. There is a direct interpretation of $d(P)$ as it pertains to its associated graph, that is, there is a k-dimensional dominance drawing of $G(P)$ if and only if $d(P) = k$. Thus the notions of the partial order dimension and the dominance drawing dimension are equivalent.

Hiraguchi [8] showed that $d(P) \leq |X|/2$, whilst Yannakakis [15] proved that it is NP-complete to decide whether the dimension of a partial order is at most k, for $k \geq 3$.

Necessary and sufficient conditions for $d(P) = 2$ are given by [5]. Let $\overline{G}(P)$ denote the complement of an undirected version of G, that is, (x, y) is an edge of $\overline{G}(P)$ if neither (x, y) nor (y, x) are edges of G. Dushnik and Miller [5] proved that $d(P) = 2$ if and only if there is a way to orient the edges of $\overline{G}(P)$ to make it a transitive digraph. Graphs that admit such an orientation of their edges are called *transitively orientable*. A polynomial time algorithm to recognize transitively orientable graphs is given in [11]. By combining the characterization in [5] and the algorithm in [11] a polynomial time algorithm is available to obtain a two dimensional dominance drawing of a transitive digraph if such a drawing is possible. An $O(\Delta|E|)$ implementation of the algorithm in [11] is described in [7], where Δ represents the maximum degree in the graph. For the case of bipartite graphs this translates to an $O(n^3)$ algorithm to determine if $d(G) \leq 2$ because the complement of any bipartite graph with n vertices always has $\Delta \in O(n)$ and $|E| \in O(n^2)$. In [12] an $O(|V|^2)$ algorithm is given to recognize two dimensional partial orders.

Kameda [9] defines a class of graph that always admits a two dimensional dominance drawing and gives an algorithm that is proportional to the size of the input graph to obtain the drawing. Kameda shows that planar DAGs with all sources and sinks embedded on the same external face so that the sources and sinks divide the boundary of this face into two contiguous parts always admit a two dimensional dominance drawing. In [14] Tamassia observes that Kameda's algorithm can be applied to the so called *reduced planar st-graphs*. In [14] straight line planar dominance drawings are obtained from DAGs with no transitive edges and with a single source and a single sink.

Another class of graph that admits a two dimensional dominance drawing is the class of *series-parallel digraphs* [2]. A series parallel digraph is recursively defined as, a single edge joining two vertices, and if G_1 and G_2 are two series parallel digraphs then they can be composed in series by unifying the source of G_2 with the sink of G_1 or composed in parallel by unifying the source of G_1 with the source of G_2 and unifying the sink of G_1 with the sink of G_2. In [2] efficient area layouts of two dimensional dominance drawings series parallel digraphs are given.

An *upward drawing* of a digraph is a planar drawing with the additional constraint that if the vertex y is reachable from the vertex x, then y appears above x in the drawing. This is similar to the dominance drawing property, with dominance required in only one of the coordinates. In [1] it is shown that transitive bipartite digraphs admit an upward drawing if and only if the graph is planar.

In [15] it is proved that deciding whether the dominance drawing dimension of a transitive bipartite graph is at most four is NP-complete. We have shown that to determine whether the dominance drawing dimension of a transitive bipartite graph is at most two is polynomial. However, it is not known whether recognizing transitive bipartite graphs with dominance drawing dimension at most three is polynomial or NP-complete. Another interesting open problem is to obtain more efficient algorithms for recognizing graphs with $d(G) \leq 2$ in the general case.

References

[1] G. Di Battista, W.-P. Liu, and I. Rival, *Bipartite Graphs, Upward Drawings, and Planarity,* Information Processing Letters, vol. 36 (1990) pp. 317-322.

[2] P. Bertolazzi, R.F. Cohen, G. Di Battista, R. Tamassia, and I.G. Tollis, *How to draw a series-parallel digraph,* Tech. Report UTDCS-9-92, The University of Texas at Dallas, (1992)

[3] K. Booth and G. Leuker, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms* J. Comput. and Sys. Sci., 13 (1976) pp. 335-379.

[4] G. Di Battista, R. Tamassia, and I.G. Tollis, *Area requirement and symmetry display in drawing graphs,* Discrete & Computational Geometry, Vol. 7, No. 4 (1992) pp. 381-401.

[5] B. Dushnik and E. W. Miller, *Partially ordered sets,* Amer. J. Math., 63 (1941) pp. 600-610.

[6] P. Eades and R. Tamassia, *Algorithms for drawing graphs: an annotated bibliography,* Tech. Report CS-89-09, Brown Univ. 1989.

[7] M. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press (1980).

[8] T. Hiraguchi, *On the dimension of partially ordered sets,* Sci. Reports Kanazawa Univ. 1 (1951) pp. 77-94.

[9] T. Kameda, *On the vector representation of the reachability in planar directed graphs,* Information Processing Letters, Vol. 3, No. 3 (1975) pp. 75-77.

[10] D. Kelly, *The 3-irreducible partially ordered sets* Can. J. Math., Vol 24 No. 2 (1977) pp. 367-383.

[11] A. Pnueli, A Lempel, and S. Even, *Transitive orientation of graphs and identification of permutation graphs,* Canad. J. Math., 23 (1971) pp. 160-175.

[12] J. Spinrad, *On comparability and permutation graphs,* SIAM J. Comput., Vol. 14, No. 3 (1985) pp. 658-670.

[13] J. Spinrad, A Brandstadt, L. Stewart, *Bipartite Permutation Graphs,* Discrete and Applied Math, 18 (1987) pp.279-282.

[14] R. Tamassia, *Drawing algorithms for planar st-graphs,* Australasian Journal of Combinatorics, vol. 2 (1990) pp. 217-235.

[15] M. Yannakakis, *The complexity of the partial order dimension problem,* SIAM J. Alg. and Disc. Meth., Vol. 3, No. 3 (1982) pp. 351-358.