Computing the cutwidth of bipartite permutation graphs in linear time*

Pinar Heggernes[1], Pim van 't Hof[1], Daniel Lokshtanov[2], and Jesper Nederlof[1]

[1] Department of Informatics, University of Bergen,
P.O. Box 7803, N-5020 Bergen, Norway.
{pinar.heggernes,pim.vanthof,jesper.nederlof}@ii.uib.no
[2] Dept. of Computer Science and Engineering, University of California, San Diego,
9500 Gilman Drive, La Jolla, CA 92093-0404, USA.
dlokshtanov@cs.ucsd.edu

Abstract. The problem of determining the cutwidth of a graph is a notoriously hard problem which remains NP-complete under severe restrictions on input graphs. Until recently, non-trivial polynomial-time cutwidth algorithms were known only for subclasses of graphs of bounded treewidth. Very recently, Heggernes et al. (SIAM J. Discrete Math., 25 (2011), pp. 1418–1437) initiated the study of cutwidth on graph classes containing graphs of unbounded treewidth, and showed that a greedy algorithm computes the cutwidth of threshold graphs. We continue this line of research and present the first polynomial-time algorithm for computing the cutwidth of bipartite permutation graphs. Our algorithm runs in linear time. We stress that the cutwidth problem is NP-complete on bipartite graphs and its computational complexity is open even on small subclasses of permutation graphs, such as trivially perfect graphs.

1 Introduction

A large variety of problems in many different domains can be formulated as graph layout problems [9]. A well known problem of this type is *cutwidth*. Given a graph G and a positive integer k, the cutwidth problem is to decide whether there is an ordering of the vertices of G such that any line inserted between two consecutive vertices in the ordering cuts at most k edges of the graph. The cutwidth of the input graph is the smallest integer for which the question can be answered positively. This problem was first proposed as a model to minimize the number of channels in a circuit [1, 16], and later it has found applications in areas like protein engineering [3], network reliability [14], automatic graph drawing [18], information retrieval [4], and as a subroutine in the cutting plane algorithm for TSP [13].

Just like numerous other graph problems of practical interest, cutwidth is NP-complete [10], even when input graphs are restricted to planar graphs of maximum degree 3 [17], split graphs [12], unit disk graphs, partial grids [8], and consequently bipartite graphs. There is a polynomial-time $O(\log^2 n)$-approximation algorithm for general graphs [15], and a polynomial-time constant factor approximation algorithm for dense graphs [2].

The knowledge on polynomial-time algorithms for the exact computation of cutwidth on restricted inputs is very limited. Cutwidth of certain trivial graph classes, like meshes or complete p-partite graphs, can be computed easily as there exist closed formulas for their cutwidth [9]. Cutwidth of proper interval graphs has a trivial solution following an interval ordering of the

* This work is supported by the Research Council of Norway and by EPSRC UK grant EP/D053633/1. An extended abstract of this paper has been presented at the 36th International Workshop on Graph Theoretic Concepts in Computer Science (WG 2010) [11].

vertices [23]. However, there are very few graph classes whose cutwidth is non-trivially computable in polynomial time. Until recently, polynomial-time cutwidth algorithms were known only for subclasses of graphs of bounded treewidth. In particular, Yannakakis [22] gave a sophisticated and technical algorithm for trees (see also [7]). Furthermore, Thilikos et al. gave an algorithm for computing the cutwidth of bounded cutwidth graphs [20], and extended this result to graphs of bounded treewidth and maximum degree [21]. The study of cutwidth on graph classes containing graphs of unbounded treewidth was initiated in [12], resulting in a linear-time algorithm for computing the cutwidth of threshold graphs. This is a simple greedy algorithm that can be applied on arbitrary graphs; it outputs the correct cutwidth if the input graph is threshold. However, even on chain graphs, which are bipartite graphs very closely related to threshold graphs, the algorithm does not compute the cutwidth.

In this paper, we show that the cutwidth of a bipartite permutation graph can be computed in linear time. As mentioned above, the cutwidth problem is NP-complete on bipartite graphs, and its computational complexity is open on permutation graphs. Thus bipartite permutation graphs are natural candidates for studying the computational complexity of the cutwidth problem. In contrast to the threshold algorithm [12] mentioned above, our algorithm cannot be applied on arbitrary graphs; it is specifically tailored to deal with bipartite permutation input graphs, and can only be run on such graphs. The correctness of our algorithm relies heavily on a characterization of bipartite permutation graphs by strong orderings [19]. Bipartite permutation graphs and threshold graphs are two unrelated subclasses of permutation graphs; the intersection of these two graph classes is restricted to stars. The class of bipartite permutation graphs properly contains the class of chain graphs mentioned above; prior to our result, no polynomial-time algorithm was known for computing the cutwidth even for chain graphs. Due to our result, bipartite permutation graphs form the first graph class of unbounded clique-width [6] whose cutwidth can be computed in polynomial time.

2 Preliminaries and notation

We consider undirected finite graphs with no loops or multiple edges. For a graph $G = (V, E)$, with vertex set V and edge set E, we define $n = |V|$ and $m = |E|$. Let $S \subseteq V$. The subgraph of G induced by S is denoted by $G[S]$. We write $G-S$ to denote the graph $G[V \setminus S]$, and we simply write $G-v$ instead of $G-\{v\}$ in case $S = \{v\}$. For two vertices $u, v \in V$ with $uv \notin E$, we write $G+uv$ to denote the graph $(V, E \cup \{uv\})$. The set of *neighbors* (or the *neighborhood*) of a vertex x of G is $N(x) = \{v \mid xv \in E\}$. The *degree* of x is $d(x) = |N(x)|$. A graph is *connected* if there is a path between any pair of its vertices. A *connected component* of a disconnected graph is a maximal connected subgraph of it.

A graph G with vertex set $\{1, \ldots, n\}$ is a *permutation* graph if there exists a permutation π of $\{1, \ldots, n\}$ such that any two vertices i and j with $i < j$ are adjacent in G if and only if $\pi(i) > \pi(j)$. A *bipartite* graph is a graph whose vertex set can be partitioned into two independent sets, called *color classes*. We write $G = (A, B, E)$ to denote a bipartite graph with color classes A and B. The partition of the vertex set into color classes of a connected bipartite graph is unique, up to symmetry. Vertices of A and of B are called *A-vertices* and *B-vertices*, respectively. We say that a vertex is *bipartite universal* if it is adjacent to all the vertices of the opposite color class.

An *ordering* of a set A is a one-to-one mapping $\sigma : A \leftrightarrow \{1, \ldots, |A|\}$. We also use the notation $\sigma = \langle a_1, a_2, \ldots, a_{|A|} \rangle$, meaning that $\sigma(a_i) < \sigma(a_j)$ when $i < j$, where each a_i is a distinct element of A, for $1 \leq i \leq |A|$. Integers $1, 2, \ldots, |A|$ are called the *positions* of σ, and $\sigma(a)$ is the *position* of a in σ. Intuitively, we will refer to the end of the ordering with a_1 as *the left* and the end of the ordering with $a_{|A|}$ as *the right*. For two elements a and a' of A, we say

that *a appears before* (or *to the left of*) a' in σ, denoted $a \prec_\sigma a'$, if $\sigma(a) < \sigma(a')$. If $\sigma(a) > \sigma(a')$, then we say that *a appears after* (or *to the right of*) a' in σ and write $a \succ_\sigma a'$. We will also use the notion of a *leftmost*, *rightmost*, and *middle* vertex or neighbor, analogously and intuitively. We say that k elements of A are *consecutive* in σ if they occupy positions $i+1, \ldots, i+k$, for some $i \in \{0, \ldots, |A|-k\}$. When we say that we *delete* an element a of A from σ, we get a new ordering in which all elements before a in σ keep their original positions, and the position of each element after a decreases by 1. We denote the new ordering by $\sigma - a$. For any subset of $A' \subseteq A$, we write $\sigma - A'$ to denote the ordering obtained from σ by consecutively deleting all the elements of A' from σ.

A *layout* of a graph $G = (V, E)$ is an ordering of V. We write $\Phi(G)$ to denote the set of all layouts of G. The *rank* of a vertex v with respect to a layout φ, denoted $rank_\varphi(v)$, is the number of neighbors of v appearing after v in φ minus the number of neighbors of v appearing before v in φ, i.e., $rank_\varphi(v) = |\{w \in N(v) \mid v \prec_\varphi w\}| - |\{w \in N(v) \mid w \prec_\varphi v\}|$. Note that the rank of a vertex can be negative. Given layout φ of a graph G and an integer $1 \leq i \leq n$, we define $L(i, \varphi, G) = \{u \in V \mid \varphi(u) \leq i\}$ and $R(i, \varphi, G) = \{u \in V \mid \varphi(u) > i\}$. The *ith gap* of φ is between $L(i, \varphi, G)$ and $R(i, \varphi, G)$, or equivalently, between positions i and $i+1$ of φ. For any set $S \subseteq V$, we define the *cut* of S to be $\theta(S, G) = \{uv \in E \mid u \in S \wedge v \notin S\}$. The *cut of G at the ith gap* of φ is defined as $\theta(i, \varphi, G) = \{uv \in E \mid u \in L(i, \varphi, G) \wedge v \in R(i, \varphi, G)\}$. Note that by definition $\theta(i, \varphi, G) = \theta(L(i, \varphi, G), G)$. We call an edge set $\theta \subseteq E$ a *cut* of φ if $\theta = \theta(i, \varphi, G)$ for some $i \in \{1, 2, \ldots n-1\}$. The *size* of a cut θ is $|\theta|$. The *cutwidth of a layout* φ of G is $\mathrm{cw}_\varphi(G) = \max_{1 \leq i \leq n} |\theta(i, \varphi, G)|$. A cut $\theta(i, \varphi, G)$ with $|\theta(i, \varphi, G)| = \mathrm{cw}_\varphi(G)$ is called a *worst cut* of φ. The *cutwidth* of G is the minimum cutwidth over all layouts of G, i.e., $\mathrm{cw}(G) = \min_{\varphi \in \Phi(G)} \mathrm{cw}_\varphi(G)$. An *optimal layout* of G is a layout φ such that $\mathrm{cw}(G) = \mathrm{cw}_\varphi(G)$. The cutwidth of a graph G equals the maximum cutwidth over all connected components of G.

As the name already indicates, bipartite permutation graphs are permutation graphs that are bipartite. The study of bipartite permutation graphs was initiated by Spinrad et al. in [19]. They present two characterizations of bipartite permutation graphs, leading to a linear-time recognition algorithm for this class as well as polynomial-time algorithms for some NP-complete problems restricted to bipartite permutation input graphs. A *strong ordering* (σ_A, σ_B) of a bipartite graph $G = (A, B, E)$ consists of an ordering σ_A of A and an ordering σ_B of B such that for all $ab, a'b' \in E$, where $a, a' \in A$ and $b, b' \in B$, $a \prec_{\sigma_A} a'$ and $b' \prec_{\sigma_B} b$ implies that $ab' \in E$ and $a'b \in E$. An ordering σ_A of A has the *adjacency property* if, for every $b \in B$, $N(b)$ consists of vertices that are consecutive in σ_A. The ordering σ_A has the *enclosure property* if, for every pair b, b' of vertices of B with $N(b) \subseteq N(b')$, the vertices of $N(b') \setminus N(b)$ appear consecutively in σ_A, implying that b is adjacent to the leftmost or the rightmost neighbor of b' in σ_A.

Theorem 1 ([19]). *The following statements are equivalent for any bipartite graph $G = (A, B, E)$.*

1. *G is a bipartite permutation graph.*
2. *G has a strong ordering.*
3. *There exists an ordering of A which has the adjacency property and the enclosure property.*

A strong ordering of a bipartite permutation graph can be computed in linear time [19]. If the graph G in Theorem 1 is connected, then it follows from the proof of Theorem 1 in [19] that we can combine statements 2 and 3 in Theorem 1 as follows.

Lemma 1 ([19]). *Let (σ_A, σ_B) be a strong ordering of a connected bipartite permutation graph $G = (A, B, E)$. Then both σ_A and σ_B have the adjacency property and the enclosure property.*

3 Cutwidth of Bipartite Permutation Graphs

In this section we prove that the cutwidth of bipartite permutation graphs can be computed in linear time. The complete algorithm is given in the proof of Theorem 2. The main ingredient is an algorithm that we call MinCutBPG. This algorithm takes as input a connected bipartite permutation graph G and a strong ordering of G, and it outputs an optimal layout of G. We will spend most of this section describing and proving the correctness of Algorithm MinCutBPG. Before we give the algorithm, we define an operation to modify a given layout in an intuitive way. Given a layout φ of a graph, when we *move* a vertex v from position i to position j, with $i < j$, only vertices in positions from i to j are affected. We get a new layout φ' in which v gets position $\varphi'(v) = j$, the vertex x with $\varphi(x) = j$ gets position $\varphi'(x) = j - 1$, and the position of each of the other affected vertices decreases by 1, similarly. All other vertices have the same position in φ' as they had in φ. What we described is a *move toward the right*. A *move toward the left* is defined symmetrically.

3.1 Description of Algorithm MinCutBPG

We now give an outline of Algorithm MinCutBPG, which takes as input a connected bipartite permutation graph $G = (A, B, E)$ and a strong ordering (σ_A, σ_B) of G. It outputs an optimal layout φ of G. After the description of the algorithm, we present its pseudocode. The three pictures in Figure 1 illustrate how the algorithm works, and we will refer to that figure in the description below.

Let $A = \{a_1, \ldots, a_s\}$ where $a_1 \prec_{\sigma_A} \cdots \prec_{\sigma_A} a_s$, and let $B = \{b_1, \ldots, b_t\}$ where $b_1 \prec_{\sigma_B} \cdots \prec_{\sigma_B} b_t$. The vertices of A will appear in the final layout φ in the same order as they appear in σ_A. Similarly, the order in which the vertices of B appear in φ corresponds to the order in which they appear in σ_B. Before deciding where the vertices of A will appear in φ with respect to the vertices of B, the algorithm first assigns the vertices of B to *boxes*. There are two types of boxes: a box X_i for every vertex $a_i \in A$, and a box $X_{i,i+1}$ for every pair of consecutive vertices $a_i, a_{i+1} \in A$. Recall that the neighbors of any vertex $b \in B$ appear consecutively in σ_A by Lemma 1. If b has even degree and its two middle neighbors are a_i and a_{i+1}, then b is assigned to box $X_{i,i+1}$. If b has odd degree and its middle neighbor is a_i, then b is assigned to box X_i. In the top picture of Figure 1, the vertices of B have been assigned to the appropriate boxes. Observe that some boxes might be empty and that the collection of non-empty boxes is a partition of B. For convenience, we also define the boxes $X_{0,1} = \emptyset$ and $X_{s,s+1} = \emptyset$.

The following observation is a direct consequence of Lemma 1, the properties of a strong ordering, and the definition of boxes.

Observation 1 *Given a connected bipartite permutation graph $G = (A, B, E)$ with $|A| = s$ and a strong ordering (σ_A, σ_B), where $\sigma_A = \langle a_1, a_2, \ldots, a_s \rangle$, let boxes $X_{0,1}, X_1, X_{1,2}, \ldots, X_s, X_{s,s+1}$ be defined as above. Then we have the following:*

1. *every vertex of X_i appears before every vertex of $X_{i,i+1}$ in σ_B, and every vertex of $X_{i,i+1}$ appears before every vertex of X_{i+1} in σ_B, for $1 \leq i \leq s$;*
2. *$N(b) = N(b')$ for any two vertices b and b' appearing in the same box.*

The algorithm now generates a layout φ' of G in which a_1 is placed first, vertices of X_1 are placed in the immediately following positions, vertices of $X_{1,2}$ are placed in the next positions, then a_2 is placed, followed by vertices of X_2, $X_{2,3}$, $\{a_3\}$, X_3, ..., $\{a_{s-1}\}$, X_{s-1}, $X_{s-1,s}$, $\{a_s\}$, and X_s. Within each box, the vertices of B belonging to that box are ordered according to σ_B. For $1 \leq i \leq s$, a_i appears just before the vertices of box X_i. The layout φ' of the graph in the middle picture of Figure 1 is obtained this way, using the boxes drawn in the top picture.

To define and obtain the final layout φ, we just need to move each a_i to its final position. This will be one of the positions of $\{a_i\} \cup X_i$ in φ'. As a consequence, we can observe already now that, for every $b \in B$, $rank_\varphi(b) \in \{-1, 0, 1\}$. The ranks of the A-vertices might have a larger range of values. Let i be any index satisfying $1 \leq i \leq s$. Recall that $rank_\varphi(a_i)$ depends on the position where a_i is placed: the further to the left a_i appears, the higher its rank. The algorithm moves a_i in such a way that $rank_\varphi(a_i)$ is as close to 0 as possible, i.e., the value of $|rank_\varphi(a_i)|$ is as small as possible, subject to the condition that the position of a_i is one of the initial positions of $\{a_i\} \cup X_i$. This is done in the following way. Note first that the set of possible positions for a_i does not intersect with the set of possible positions for any other A-vertex a_j with $i \neq j$. Furthermore, $rank_\varphi(a_i)$ is only dependent on the neighbors of a_i and no two A-vertices are adjacent. Therefore, the placement of each a_i among the positions of $\{a_i\} \cup X_i$ can be decided independently of the placements of the other A-vertices. By Lemma 1, the neighbors of a_i appear consecutively in σ_B. If a_i has odd degree, then let b be the middle neighbor of a_i in σ_B. If a_i has even degree, then let b be the right one of the two middle neighbors of a_i. If $b \in X_i$, then we move a_i to the position just before the position of b. If b appears in a box to the left of X_i, then we do not move a_i. If b appears in a box to the right of X_i, then we move a_i to the last position among the positions of X_i. Thus, if a_i is placed between two vertices of X_i, then its rank is 0 or 1. If a_i is placed before or after all vertices of X_i, then its rank can be higher or lower. This completes the definition and computation of φ.

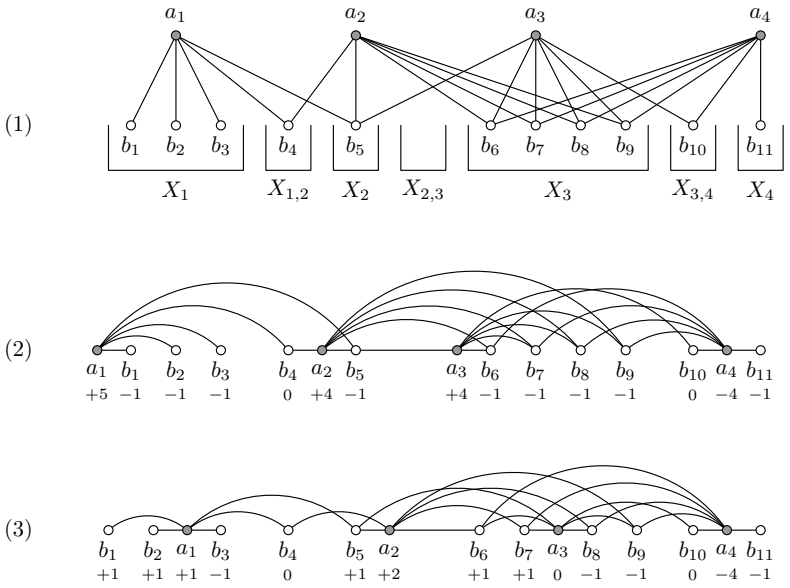

Fig. 1. An illustration of the way algorithm MinCutBPG computes an optimal layout when given as input a connected bipartite permutation graph $G = (A, B, E)$ and a strong ordering (σ_A, σ_B). (1) The vertices of B are assigned to boxes. (2) A layout φ' of G is generated by placing each vertex a_i immediately to the left of box X_i. The integer below each vertex indicates its rank with respect to φ'. (3) The optimal layout φ is obtained by placing the vertex a_i in one of the initial positions of $\{a_i\} \cup X_i$ such that its rank is as close to 0 as possible.

We make the following observation about the layout φ generated by Algorithm MinCutBPG, which is a direct consequence of Lemma 1.

Observation 2 *Let $G = (A, B, E)$ be a connected bipartite permutation graph and let (σ_A, σ_B) be a strong ordering of G, where $\sigma_A = \langle a_1, a_2, \ldots, a_s \rangle$. Let φ be the layout of G generated by Algorithm* MinCutBPG *on input G and (σ_A, σ_B). Then, for every $1 \leq i \leq s$, we have the following:*

1. *for any $b \in X_{i,i+1}$, $rank_\varphi(b) = 0$;*
2. *for any $b \in X_i$, $rank_\varphi(b) = 1$ if $b \prec_\varphi a_i$ and $rank_\varphi(b) = -1$ if $a_i \prec_\varphi b$;*
3. *every $b \in X_{i-1,i} \cup X_i \cup X_{i,i+1}$ is adjacent to a_i.*

To conclude this section, we give pseudocode for the algorithm MinCutBPG.

Algorithm MinCutBPG

input: a connected bipartite permutation graph $G = (A, B, E)$ and a strong ordering (σ_A, σ_B) of G;
output: an optimal layout φ of G;

let $\sigma_A = \langle a_1, a_2, \ldots, a_s \rangle$;
let $\sigma_B = \langle b_1, b_1, \ldots, b_t \rangle$;
initialize $X_{0,1}, X_1, X_{1,2}, \ldots, X_s, X_{s,s+1}$ to be empty;
for $i = 1$ **to** t **do**
    if $|N(b_i)|$ is even with middle neighbors a_j, a_{j+1} **then**
        $X_{j,j+1} = X_{j,j+1} \cup \{b_i\}$;
    else
        let a_j be the unique middle neighbor of b_i;
        $X_j = X_j \cup \{b_i\}$;
$k = 1$;
for $i = 1$ **to** s **do**
    $\varphi(a_i) = k$;
    $k = k + 1$;
    place the vertices of X_i in positions $k, \ldots, k + |X_i| - 1$ of φ in the order they appear in σ_B;
    $k = k + |X_i|$;
    place the vertices of $X_{i,i+1}$ in positions $k, \ldots, k + |X_{i,i+1}| - 1$ of φ in the order they appear in σ_B;
    $k = k + |X_{i,i+1}|$;
for $i = 1$ **to** s **do**
    $q = |N(a_i) \cap L(\varphi(a_i), \varphi, G)|$;
    $p = \lfloor d(a_i)/2 \rfloor$;
    if $p > q + |X_i|$ **then**
        move a_i to the position of the rightmost vertex of X_i in φ;
    else if $q < p \leq q + |X_i|$ **then**
        move a_i to the $(p - q)$th position among the positions of X_i in φ;

3.2 Correctness of Algorithm MinCutBPG

We show that Algorithm MinCutBPG produces an optimal layout when the input is a connected bipartite permutation graph and a strong ordering of that graph. We assume for contradiction that there is a connected bipartite permutation graph G for which the algorithm outputs a layout φ such that $\mathrm{cw}_\varphi(G) > \mathrm{cw}(G)$. Such a graph is called a *counterexample*, and we write $\mathcal{G}$ to denote the set of all counterexamples. Let $\mathcal{G}' \subseteq \mathcal{G}$ be the set of counterexamples having

the minimum number of vertices among all counterexamples, and let $\mathcal{G}'' \subseteq \mathcal{G}'$ be the set of graphs in $\mathcal{G}'$ having the maximum number of edges among all graphs in $\mathcal{G}'$. A graph in $\mathcal{G}''$ is called a *tight counterexample*. If there exists a counterexample, then there also exists a tight counterexample.

For the statements and the proofs of the following lemmas, let $G = (A, B, E)$ with $E \neq \emptyset$ be a connected bipartite permutation graph that is a tight counterexample, and let (σ_A, σ_B) be a strong ordering of G such that $\sigma_A = \langle a_1, \ldots, a_s \rangle$ and $\sigma_B = \langle b_1, \ldots, b_t \rangle$. Furthermore, let $\varphi = \langle v_1, \ldots, v_n \rangle$ be the layout of G generated by Algorithm MinCutBPG on input G and (σ_A, σ_B).

Lemma 2. *Let $\theta(j, \varphi, G)$ be a worst cut of φ. Then we have the following:*

1. *a_1 is adjacent to the rightmost B-vertex of $L(j, \varphi, G)$;*
2. *b_1 is adjacent to the rightmost A-vertex of $L(j, \varphi, G)$;*
3. *a_s is adjacent to the leftmost B-vertex of $R(j, \varphi, G)$;*
4. *b_t is adjacent to the leftmost A-vertex of $R(j, \varphi, G)$.*

Proof. We first prove claim 1. Let $\theta = \theta(j, \varphi, G)$ and let b be the rightmost B-vertex of $L = L(j, \varphi, G)$. If $b \prec_\varphi a_1$, then all B-vertices in L appear before a_1 in φ, and b is the vertex just before a_1 in φ, implying that $\varphi(b) = \varphi(a_1) - 1$. Hence $b \in X_1$, and by Observation 2, $a_1 b \in E$. Now assume that $a_1 \prec_\varphi b$, and suppose for contradiction that a_1 is not adjacent to b. Note that this means that $b \notin X_1$, since every vertex in box X_1 is adjacent to a_1 by Observation 2. We claim that $G' = G - (\{a_1\} \cup X_1)$ is a counterexample, contradicting the assumption that G is a tight counterexample. Observe that G' is a connected bipartite permutation graph and $(\sigma_A - a_1, \sigma_B - X_1)$ is a strong ordering of G'. We will prove the claim by showing that θ is a cut of the layout φ' returned by Algorithm MinCutBPG on input G' and $(\sigma_A - a_1, \sigma_B - X_1)$. Since $a_1 b \notin E$ and G is connected, b has a neighbor a_i for some $i \geq 2$. Then a_1 has no neighbors in $R = R(j, \varphi, G)$ as a result of the properties of a strong ordering. None of the vertices in X_1 has a neighbor in R either, because they are adjacent to a_1 only. Therefore, θ is a cut of $\varphi - (\{a_1\} \cup X_1)$. We will show that all vertices of $L \setminus (\{a_1\} \cup X_1)$ that appear to the left of b in $\varphi - (\{a_1\} \cup X_1)$ also appear to the left of b in φ'.

Clearly, the relative orderings of the A-vertices and of the B-vertices are the same in φ' as in φ. Let us analyze how the deletion of the vertices in $\{a_1\} \cup X_1$ can affect the ranks of vertices and the boxes that they belong to. Deleting $\{a_1\} \cup X_1$ does not change the rank of b, or any A-vertex, or any B-vertex in $R(j, \varphi, G)$, since none of these vertices is adjacent to any of the vertices in $\{a_1\} \cup X_1$. Consequently, b appears in the same box after the deletion of $\{a_1\} \cup X_1$ as it did before, and so do all the B-vertices to the right of b. Let $a \neq a_1$ be the rightmost A-vertex of L; note that a might not be defined in case a_1 is the only A-vertex of L. Either a or b is the rightmost vertex of L in φ. In either case, since the ranks of a and b did not change, a and b have the same relative order to each other in φ' as in φ. The only vertices whose ranks might change by the deletion of $\{a_1\} \cup X_1$ are the B-vertices of L that were adjacent to a_1. However, these vertices cannot appear to the right of b in φ', as the algorithm respects the strong ordering $(\sigma_A - a_1, \sigma_B - X_1)$. As a result, the set of vertices that appear to the left of b is the same in φ' as in φ. Since all the vertices to the right of b appear in exactly the same positions as before, θ is a cut of φ'. Since $\mathrm{cw}(G') \leq \mathrm{cw}(G)$ and the size of the cut did not change, we conclude that G' is a counterexample with at least one fewer vertex than G, giving us the desired contradiction. This completes the proof of claim 1.

The proof of claim 2 strongly resembles the proof of claim 1, but is slightly easier. If a_1 is the rightmost A-vertex of L, then claim 2 holds, as b_1 is adjacent to a_1 due to the assumption that G is connected. Let a be the rightmost A-vertex of L, and assume that $a \neq a_1$. Suppose for contradiction that b_1 is not adjacent to a. We claim that $G' = G - \{b_1\}$ is a counterexample,

contradicting the assumption that G is a tight counterexample. Note that G' is connected, and that $(\sigma_A, \sigma_B - \{b_1\})$ is a strong ordering of G'. Since b_1 is not adjacent to a, vertex b_1 is not adjacent to any A-vertex in R. Clearly, b_1 is not adjacent to any B-vertex either. Hence, the only vertices whose ranks might be affected by the deletion of b_1 are the A-vertices in L that are adjacent to b_1. Just like in the proof of claim 1, we can show that $\theta = \theta(j, \varphi, G)$ is a cut of the layout φ' returned by Algorithm MinCutBPG on input G' and $(\sigma_A, \sigma_B - \{b_1\})$, yielding the desired contradiction. Since this time graph G' is obtained from G by deleting a single vertex (b_1) from G rather than a set of vertices ($\{a_1\} \cup X_1$), the details are easier and therefore omitted.

By symmetry, the correctness of claims 3 and 4 immediately follows from the correctness of claims 1 and 2, respectively.

Lemma 3. *Let $\theta(j, \varphi, G)$ be a worst cut of φ. Then both $G[L(j, \varphi, G)]$ and $G[R(j, \varphi, G)]$ are complete bipartite graphs.*

Proof. Let a and b be the rightmost A-vertex and B-vertex of $L = L(j, \varphi, G)$, respectively. By Lemma 2, a_1 is adjacent to b and b_1 is adjacent to a. By the definition of a strong ordering, a_1 is adjacent to b_1 and a is adjacent to b. Since G is connected, and σ_A and σ_B have the adjacency property by Lemma 1, a and a_1 are adjacent to all B-vertices in L, and b and b_1 are adjacent to all A-vertices in L. As a result, every vertex of $A \cap L$ is adjacent to every vertex of $B \cap L$. This means that $G[L(j, \varphi, G)]$ is complete bipartite. By symmetry the same holds for $G[R(j, \varphi, G)]$.

Lemma 4. *There is a worst cut $\theta(j, \varphi, G)$ of φ such that v_j and v_{j+1} belong to different color classes.*

Proof. Let $L = L(j, \varphi, G)$ and let $R = R(j, \varphi, G)$. Assume that either L or R, say L, contains vertices of only one color class. Since G is connected and $E \neq \emptyset$, G contains vertices from both color classes. Let us consider the smallest index $k \geq j$ such that there is a vertex of the other color class in position $k + 1$. Then $|\theta(k, \varphi, G)| \geq |\theta(j, \varphi, G)|$ because $L \subseteq L(k, \varphi, G)$, there are no edges between the vertices of $L(k, \varphi, G)$, and each vertex of $L(k, \varphi, G)$ has a neighbor in $R(k, \varphi, G)$. Hence we can conclude that there is a worst cut at the gap between two vertices of opposite color. The case where R contains only vertices of one color class is completely symmetric. For the rest of the proof, assume that both L and R contain vertices of both color classes.

Assume first that both v_j and v_{j+1} are B-vertices. Let a_i be the rightmost A-vertex in L, which means that a_{i+1} is the leftmost A-vertex in R. Both $b = v_j$ and $b' = v_{j+1}$ are between a_i and a_{i+1}; more precisely, $a_i \prec_\varphi b \prec_\varphi b' \prec_\varphi a_{i+1}$. If $rank_\varphi(b) = 1$, then $b \in X_{i+1}$ by Observation 2. Then by Observation 1, $b' \in X_{i+1}$ as well, and consequently $rank_\varphi(b') = 1$. Thus we can conclude that b and b' have the same neighborhood and they have one more neighbor in R than in L. In this case $\theta(j, \varphi, G)$ cannot be a worst cut, because the cut just to the right of b' has larger size. Therefore, $rank_\varphi(b) \leq 0$, which means that b has at least as many neighbors to the left as it has to the right. Since b has no neighbors appearing between a_i and b, the cut just to the right of a_i is of size at least $|\theta(j, \varphi, G)|$. Hence we can take that cut as the worst cut. Consequently there is a worst cut at the gap between an A-vertex and a B-vertex.

Assume now that both v_j and v_{j+1} are A-vertices, say a_i and a_{i+1}. First we show that in this case both a_i and a_{i+1} are bipartite universal. Assume for contradiction that this is not true, and let b be the leftmost B-vertex in R which is not a neighbor of a_i. We claim that $G' = G + a_i b$ is also a counterexample, contradicting the assumption that G is a tight counterexample. Recall that $G[L]$ and $G[R]$ are complete bipartite graphs due to Lemma 3.

Now observe that G' is a bipartite permutation graph and (σ_A, σ_B) is a strong ordering of G'. Let φ' be the layout computed by Algorithm MinCutBPG on input G' and (σ_A, σ_B). Let us analyze how the layout φ can change to φ' due to the addition of edge a_ib. Observe that $X_{i,i+1}$ is empty before the addition of edge a_ib, since a_i and a_{i+1} are consecutive in φ. When we add edge a_ib, vertex b gets one more neighbor to the left, and thus might appear in a box further to the left than the box it was in before. By Observation 1 we know that b was not in X_i or $X_{i,i+1}$ before the addition of edge a_ib. Now it can enter $X_{i,i+1}$ but it cannot enter X_i, since it only gained one neighbor. This means that it can move past a_{i+1} toward the left, but it cannot move past a_i. Thus $L(j, \varphi', G') = L$ and $R(j, \varphi', G') = R$, although some vertices in R might have changed positions. Consequently, $\theta(j, \varphi', G') = \theta(j, \varphi, G) \cup \{a_ib\}$ is a cut of φ', which means that φ' has a cut whose size is 1 larger than a worst cut of φ. Since $\mathrm{cw}(G') \leq \mathrm{cw}(G)+1$, G' is a counterexample, contradicting the assumption that G is a tight counterexample. Thus there cannot be a B-vertex in R that a_i is not adjacent to. By Lemma 3 we know that a_i is adjacent to all B-vertices in L, and hence a_i is bipartite universal. By symmetry and with similar arguments, a_{i+1} is also bipartite universal. This means that $rank_\varphi(a_i) = rank_\varphi(a_{i+1})$. If this rank is negative, then the cut at the $(j-1)$th gap is a larger cut than $\theta(j, \varphi, G)$, since a_i and a_{i+1} have more neighbors in L than in R. Symmetrically, if this rank is positive, then the cut at the $(j+1)$th gap is a larger cut. Therefore $rank_\varphi(a_i) = rank_\varphi(a_{i+1}) = 0$, because otherwise we get a contradiction to the assumption that $\theta(j, \varphi, G)$ is a worst cut. This means that a_i and a_{i+1} have as many neighbors in L as they have in R. Since a_i and a_{i+1} are both bipartite universal and they are not adjacent to each other, the cut at the $(j-1)$th gap and the cut at the $(j+1)$th gap have the same size as $\theta(j, \varphi, G)$. Hence we can take one of these cuts as a worst cut. We can repeat this argument until we reach a B-vertex on the other side of a worst cut.

Lemma 5. *There is a worst cut $\theta(j, \varphi, G)$ of φ such that both v_j and v_{j+1} are bipartite universal.*

Proof. By Lemma 4, we know that there is a worst cut $\theta = \theta(j, \varphi, G)$ such that v_j and v_{j+1} belong to different color classes. Let us now show that both v_j and v_{j+1} are bipartite universal. Let $a = v_j \in A$ and let $b = v_{j+1} \in B$. By Lemma 3 we know that a is adjacent to every B-vertex in L and b is adjacent to every A-vertex in R. If $ab_t \in E$, then a is bipartite universal as a result of the properties of a strong ordering. If $ab_t \notin E$, then we claim that $G' = G - b_t$ is also a counterexample, contradicting the assumption that G is a tight counterexample. We observe that G' is a bipartite permutation graph with strong ordering $(\sigma_A, \sigma_B - b_t)$. Since b_t has no neighbors in L as a result of the properties of a strong ordering, θ is a cut of $\varphi - b_t$. Let φ' be the layout computed by MinCutBPG on input G' and $(\sigma_A, \sigma_B - b_t)$. Since no B-vertex was adjacent to b_t, every remaining B-vertex appears in the same box after the deletion of b_t as it did before. However, an A-vertex a_i that was adjacent to b_t might move one position to the left inside the box X_i. Hence a_i can move past b toward the left, but it cannot move past a, since the algorithm respects the ordering σ_A. Consequently, all vertices of L to the left of a in φ also appear to the left of a in φ'. Thus θ is a cut of φ'. Since $\mathrm{cw}(G') \leq \mathrm{cw}(G)$ and the size of the cut did not change, G' is a counterexample, contradicting the assumption that G is a tight counterexample. Hence a is bipartite universal. To show that b is bipartite universal we use similar arguments: by symmetry, if $a_1b \notin E$, then $G' = G - a_1$ is a counterexample as well. Finally, the case where $v_j \in B$ and $v_{j+1} \in A$ is completely symmetric.

Corollary 1. *There is a worst cut $\theta(j, \varphi, G)$ of φ such that v_j and v_{j+1} belong to different color classes and they are both bipartite universal.*

Proof. The proof of Lemma 5 takes a cut as mentioned in Lemma 4, and shows the claim of Lemma 5 using the same cut. Hence, there is a cut that satisfies both lemmas at the same time, and the corollary follows.

Lemma 6. *There is a worst cut $\theta(j, \varphi, G)$ such that there are $\lfloor |A|/2 \rfloor$ A-vertices and $\lceil |B|/2 \rceil$ B-vertices on one side of the jth gap of φ, and there are $\lceil |A|/2 \rceil$ A-vertices and $\lfloor |B|/2 \rfloor$ B-vertices on the other side of the jth gap.*

Proof. By Corollary 1 there is a worst cut $\theta(j, \varphi, G)$ such that vertices in positions j and $j+1$ belong to different color classes and are bipartite universal. Let $b \in B$ be one of these vertices; it can be on either side of the cut. Let $L = L(j, \varphi, G)$ and $R = R(j, \varphi, G)$. By Observation 2, $rank_\varphi(b) \in \{-1, 0, 1\}$. Hence the difference between $|N(b) \cap L|$ and $|N(b) \cap R|$ is at most 1. Since b is adjacent to every A-vertex, we immediately obtain that $|A \cap L|$ and $|A \cap R|$ differ by at most 1. Consequently, there are $\lfloor |A|/2 \rfloor$ A-vertices on one side, and $\lceil |A|/2 \rceil$ A-vertices on the other side of the jth gap of φ.

Next we show that $|B \cap L|$ and $|B \cap R|$ differ by at most 1. Let a_i be the rightmost A-vertex in L. Let B' be the set of B-vertices between a_i and a_{i+1} in φ. By Corollary 1, we may assume that B' is not empty and that either $a_i = v_j$ or $a_{i+1} = v_{j+1}$. We will first show that a_i, a_{i+1} and all vertices in B' are bipartite universal. Let us first consider the case where $a_i = v_j$. Then we already know by Corollary 1 that a_i and the leftmost vertex b in B' are bipartite universal. We also know that each vertex of B' is adjacent to all A-vertices in R by Lemma 3. Let us add all the missing edges between the vertices of B' and the A-vertices in L to obtain a modified graph G'. Since b is bipartite universal in G, the new graph G' is a bipartite permutation graph, and (σ_A, σ_B) is a strong ordering of G'. We will show that G' is a counterexample as well. Let φ' be the layout computed by MinCutBPG on input G' and (σ_A, σ_B). In G', b is still bipartite universal, and therefore it stays in the box that it was in before. The position of a vertex $b' \neq b$ of B' cannot be further to the left in φ' than in φ, as b' is not allowed to pass the other vertices of B', and in particular b, toward the left, since the algorithm respects the strong ordering. For the same reason, no A-vertex can change position from one side of a_i to the other side of a_i. Therefore, $L(j, \varphi', G') = L(j, \varphi, G)$. Let ℓ be the number of edges we added to G to obtain G'. Hence, $|\theta(j, \varphi', G')| = |\theta(j, \varphi, G)| + \ell$ and $\mathrm{cw}(G') \leq \mathrm{cw}(G) + \ell$. As φ' has a worst cut larger than the cutwidth of G', G' is a counterexample, contradicting the assumption that G is a tight counterexample. Thus all vertices in B' are bipartite universal. To show that a_{i+1} is bipartite universal we use similar arguments. If it is not bipartite universal, consider the rightmost B-vertex b' in L which a_{i+1} is not adjacent to. Then $G + b'a_{i+1}$ is a bipartite permutation graph with the same strong ordering as G, and it is a counterexample as well. The case where $a_{i+1} = v_{j+1}$ is completely symmetric, since in that case we know that a_{i+1} and the rightmost vertex of B' are bipartite universal.

Now we know that a_i, a_{i+1}, and the set B' of B-vertices between them are all bipartite universal. Thus the vertices in B' all have the same neighborhood and the same rank. This means that all the vertices of B' belong to exactly one of the sets X_i, $X_{i,i+1}$, or X_{i+1}. First assume that all vertices of B' belong to $X_{i,i+1}$. Then by Observation 2, they all have rank 0. Observe that $rank_\varphi(a_i) \geq 0$, since otherwise moving the cut toward the left just to the left of a_i gives a larger cut. Similarly, $rank_\varphi(a_{i+1}) \leq 0$; otherwise moving the cut toward the right just to the right of a_{i+1} gives a larger cut. As a consequence, at least $\lceil |B|/2 \rceil$ B-vertices are to the right of a_i and at least $\lceil |B|/2 \rceil$ B-vertices are to the left of a_{i+1}. Therefore, there is a position k with $\varphi(a_i) \leq k < \varphi(a_{i+1})$ that satisfies the following: the number of B-vertices to the right of the kth gap and the number of B-vertices to the left of the kth gap differ by at most 1. Since the vertices in B' are all bipartite universal and not adjacent to each other, we can conclude that they all have the same number of neighbors in L as they have in R.

Consequently, $|\theta(k,\varphi,G)| = |\theta(j,\varphi,G)|$, and we can take $\theta(k,\varphi,G)$ as a worst cut. Thus we arrive at a situation where there are $\lfloor |B|/2 \rfloor$ B-vertices on one side of the kth gap and $\lceil |B|/2 \rceil$ B-vertices on the other side. Note that $A \cap L(k,\varphi,G) = A \cap L$ and $A \cap R(k,\varphi,G) = A \cap R$. Consequently, the number of A-vertices on either side of the worst cut remains unchanged.

It remains to study the case where all vertices of B' belong to either X_i or X_{i+1}. If they are all in X_i, then they all have rank -1 by Observation 2. In this case, the cut just to the right of a_i is larger in size than the cut just to the left of a_{i+1}, since the vertices of B' have more neighbors to the left of B' than they have to the right of B'. Therefore, $a_i = v_j$. The rank of a_i cannot be larger than 1, because if it were, then Algorithm MinCutBPG would have placed a_i further right in X_i. On the other hand, $rank_\varphi(a_i) \geq 0$, since otherwise moving the cut toward the left just to the left of a_i gives a larger cut. Hence $rank_\varphi(a_i) \in \{0,1\}$. Since a_i is adjacent to all B-vertices, this implies that there are $\lfloor |B|/2 \rfloor$ B-vertices on one side of the cut and $\lceil |B|/2 \rceil$ B-vertices on the other side. If all vertices of B' belong to X_{i+1}, then they all have rank 1 and the cut is just to the left of a_{i+1}, by the same arguments as above. As we saw before, $rank_\varphi(a_i) \leq 0$, since otherwise moving the cut toward the right just to the right of a_i gives a larger cut. If the rank of a_{i+1} were smaller than 0, then it would appear further left in X_{i+1} than it does, and therefore its rank must be 0 or 1. Exactly as above, we conclude that there are $\lfloor |B|/2 \rfloor$ B-vertices on one side of the cut and $\lceil |B|/2 \rceil$ B-vertices on the other side. Again, the number of A-vertices on either side of the cut is the same as before.

We have thus shown that there is a cut $\theta(k,\varphi,G)$ with the same size as $\theta(j,\varphi,G)$, where $\varphi(a_i) \leq k < \varphi(a_{i+1})$, such that there are at least $\lfloor |A|/2 \rfloor$ A-vertices and at least $\lfloor |B|/2 \rfloor$ B-vertices one one side of the cut, and there are at most $\lceil |A|/2 \rceil$ A-vertices and at most $\lceil |B|/2 \rceil$ B-vertices on the other side of the cut. If either $|A|$ or $|B|$ is even, then the statement of the lemma already follows from what we have proved so far. So the only problem might occur when A and B both have odd cardinality. Assume that both $|A|$ and $|B|$ are odd. Note that this implies that none of the bipartite universal vertices has rank 0. Recall that all vertices of B' between a_i and a_{i+1} have the same rank. If they all have rank 0, then there are equally many A-vertices on either side of the cut. Thus A has even cardinality, which contradicts the assumption that $|A|$ and $|B|$ are both odd. If all vertices of B' have rank -1, then there is one more A-vertex in $L' = L(k,\varphi,G)$ than in $R' = R(k,\varphi,G)$. On the other hand, we showed earlier that the rank of a_i is 0 or 1. This, together with the fact that a_i is bipartite universal and therefore cannot have rank 0, implies that there is one more B-vertex in R' than in L'. If all vertices of B' have rank 1 then, symmetrically, we must have $rank_\varphi(a_{i+1}) = -1$. Consequently there is one more A-vertex in R' than in L', and one more B-vertex in L' than in R'. Thus the statement of the lemma follows.

We are now ready to prove the main theorem of this paper.

Theorem 2. *The cutwidth of a bipartite permutation graph can be computed in linear time.*

Proof. We describe the main algorithm for computing the cutwidth of a bipartite permutation graph G. First we compute a strong ordering of each connected component of G. Then we run MinCutBPG on each connected component with the computed strong ordering of that connected component. We concatenate the returned layouts from each of these calls into one layout φ for G. The order in which the layouts are concatenated does not matter, as the cuts at the concatenation points are empty. We check every position j with $1 \leq j < n$ to find a largest cut $\theta(j,\varphi,G)$, and we output $|\theta(j,\varphi,G)|$ as the cutwidth of G. If Algorithm MinCutBPG is correct, then clearly the output of the described algorithm is equal to $\mathrm{cw}(G)$.

Before we prove the correctness of Algorithm MinCutBPG, let us analyze the running time of the above algorithm. By the results of [19], computing a strong ordering for each connected component of G takes in total $O(n+m)$ time. The running time of Algorithm MinCutBPG

is also $O(n+m)$. To see this, observe that in the first loop, when deciding the box of a B-vertex, we never need to consider boxes to the left of the most recently considered box. By Observation 1, the next B-vertex is placed in either the box in which the previous B-vertex was placed, or a box further to the right. Thus running MinCutBPG on each connected component takes $O(n+m)$ time for the whole graph. Concatenating the returned layouts and finding the largest cut takes $O(n+m)$ time, and the overall running time follows.

Let us prove that Algorithm MinCutBPG correctly computes the cutwidth of a connected bipartite permutation graph. Assume for contradiction that there is a tight counterexample $G = (A, B, E)$. By Lemma 6, we know that there is a worst cut $\theta = \theta(j, \varphi, G)$ of the layout φ computed by Algorithm MinCutBPG on G, such that there are $\lfloor |A|/2 \rfloor$ A-vertices and $\lceil |B|/2 \rceil$ B-vertices on one side of the jth gap of φ, and $\lceil |A|/2 \rceil$ A-vertices and $\lfloor |B|/2 \rfloor$ B-vertices on the other side. Let $F = \{ab \notin E \mid a \in A \wedge b \in B\}$ be the set of missing edges in G. Then $F \cap E = \emptyset$ and $(A, B, (E \cup F))$ is a complete bipartite graph. Since by Lemma 3 vertices on either side of θ induce a complete bipartite graph, we have that for each $ab \in F$, a and b are on different sides of θ. Thus we can conclude the following about the size of θ:

$$|\theta| = \left\lfloor \frac{|A|}{2} \right\rfloor \left\lfloor \frac{|B|}{2} \right\rfloor + \left\lceil \frac{|A|}{2} \right\rceil \left\lceil \frac{|B|}{2} \right\rceil - |F| \ .$$

Let S be *any* set of $\lfloor |A|/2 \rfloor + \lceil |B|/2 \rceil$ vertices of G. We claim that $|\theta(S, G)| \geq |\theta|$, regardless of how many A-vertices and how many B-vertices there are in S. To consider all possibilities, let there be $\lfloor |A|/2 \rfloor - x$ A-vertices and $\lceil |B|/2 \rceil + x$ B-vertices in S, for an appropriate (positive, zero or negative) integer x. Consequently, there are $\lceil |A|/2 \rceil + x$ A-vertices and $\lfloor |B|/2 \rfloor - x$ B-vertices in $(A \cup B) \setminus S$. Some of the set F of missing edges might have endpoints on different sides of the cut $\theta(S, G)$ and some might not. Since $(A, B, (E \cup F))$ is a complete bipartite graph, we know the following about the size of $\theta(S, G)$:

$$|\theta(S,G)| \geq \left(\left\lfloor \frac{|A|}{2} \right\rfloor - x \right) \left(\left\lfloor \frac{|B|}{2} \right\rfloor - x \right) + \left(\left\lceil \frac{|A|}{2} \right\rceil + x \right) \left(\left\lceil \frac{|B|}{2} \right\rceil + x \right) - |F|$$

$$= \left\lfloor \frac{|A|}{2} \right\rfloor \left\lfloor \frac{|B|}{2} \right\rfloor - \left\lfloor \frac{|A|}{2} \right\rfloor x - \left\lfloor \frac{|B|}{2} \right\rfloor x + x^2 + \left\lceil \frac{|A|}{2} \right\rceil \left\lceil \frac{|B|}{2} \right\rceil + \left\lceil \frac{|A|}{2} \right\rceil x + \left\lceil \frac{|B|}{2} \right\rceil x + x^2 - |F|$$

$$= |\theta| + 2x^2 + x \left(\left\lceil \frac{|A|}{2} \right\rceil - \left\lfloor \frac{|A|}{2} \right\rfloor + \left\lceil \frac{|B|}{2} \right\rceil - \left\lfloor \frac{|B|}{2} \right\rfloor \right) .$$

Note that the value of the expression in parentheses in the last line of the equation is 0, 1, or 2. Consequently, for all possible values of x, we have that $|\theta(S, G)| \geq |\theta|$.

Let φ^* be an optimal layout of G, and let $j = \lfloor |A|/2 \rfloor + \lceil |B|/2 \rceil$. Let $S^* = L(j, \varphi^*, G)$. Hence S^* contains $\lfloor |A|/2 \rfloor + \lceil |B|/2 \rceil$ vertices and $\theta(j, \varphi^*, G) = \theta(S^*, G)$. Clearly $\mathrm{cw}(G) \geq |\theta(j, \varphi^*, G)| = |\theta(S^*, G)|$. However, for any such set S^*, we have shown above that a worst cut θ of the layout computed by Algorithm MinCutBPG has the property $|\theta(S^*, G)| \geq |\theta|$. Therefore, $\mathrm{cw}(G) \geq |\theta|$, contradicting the assumption that G is a counterexample. Consequently, no counterexample exists, and the algorithm correctly computes the cutwidth of every connected bipartite permutation graph.

4 Concluding Remarks

Algorithm MinCutBPG takes as input a connected bipartite permutation graph $G = (A, B, E)$ and a strong ordering (σ_A, σ_B) of G. Before the algorithm is called, $O(n+m)$ time is spent on recognizing G as a bipartite permutation graph and computing a strong ordering of G. Within

the same running time one can assign two integers $\ell(v)$ and $r(v)$ to every vertex $v \in A \cup B$ for the following purpose. If $v \in A$, then $\ell(v)$ and $r(v)$ are the positions of the leftmost and the rightmost neighbor of v in σ_B. If $v \in B$, then $\ell(v)$ and $r(v)$ are the positions of the leftmost and the rightmost neighbor of v in σ_A. Observe that with this information, $d(v)$ can be computed in constant time, and the middle neighbor of a vertex can be found in constant time. Consequently, if $\ell(v)$ and $r(v)$ are supplied to MinCutBPG as input for every $v \in A \cup B$, the running time of MinCutBPG is in fact $O(n)$.

Within the same running time, we can also modify MinCutBPG so that it outputs $d_L(v)$ and $d_R(v)$ for each vertex v of G, the number of neighbors of each vertex appearing to its left and to its right, respectively, in the final layout φ generated by the algorithm. These two values can be computed in constant time for each vertex at the end of each iteration of the last **for**-loop, and hence in $O(n)$ time in total. Using this information, the cutwidth of the produced optimal layout, and consequently the cutwidth of any bipartite permutation graph, can be computed in $O(n)$ time as well.

With our results in addition to the results of [12], the cutwidth of two unrelated subclasses of permutation graphs can be computed in linear time: threshold graphs and bipartite permutation graphs. We leave as an open problem to decide the computational complexity of computing the cutwidth of permutation graphs. In fact, it would be interesting to know the computational complexity of cutwidth on other well known subclasses of permutation graphs, like cographs or even their subclass trivially perfect graphs.

References

1. D. Adolphson and T. C. Hu, *Optimal linear ordering,* SIAM J. Appl. Math., 25 (1973), pp. 403–423.
2. S. Arora, A. Frieze, and H. Kaplan, *A new rounding procedure for the assignment problem with applications to dense graphs arrangements,* In Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 1996, pp. 21–30.
3. G. Blin, G. Fertin, D. Hermelin, and S. Vialette, *Fixed-parameter algorithms for protein similarity search under RNA structure constraints,* In Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG), Lecture Notes in Comput. Sci. 3787, Springer, Berlin, 2005, pp. 271–282.
4. R. A. Botafogo, *Cluster analysis for hypertext systems,* In Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, ACM, 1993, pp. 116–125.
5. A. Brandstädt, V. B. Le, and J. Spinrad, *Graph Classes: A Survey.* SIAM, Philadelphia, 1999.
6. A. Brandstädt and V. V. Lozin, *On the linear structure and clique-width of bipartite permutation graphs,* Ars Combin., 67 (2003), pp. 273–289.
7. M. J. Chung, F. Makedon, I. H. Sudborough, and J. Turner, *Polynomial time algorithms for the min cut problem on degree restricted trees,* In Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 1982, pp. 262–271.
8. J. Díaz, M. Penrose, J. Petit, and M. J. Serna, *Approximating layout problems on random geometric graphs,* J. Algorithms, 39 (2001), pp. 78–117.
9. J. Díaz, J. Petit, and M. J. Serna, *A survey of graph layout problems,* ACM Computing Surveys, 34 (2002), pp. 313–356.
10. F. Gavril, *Some NP-complete problems on graphs,* In Proceedings of the 11th Conference on Information Sciences and Systems, Johns Hopkins University, Baltimore, 91–95, 1977.
11. P. Heggernes, P. van 't Hof, D. Lokshtanov, and J. Nederlof, *Computing the cutwidth of bipartite permutation graphs in linear time,* In Proceedings of the 36th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), Lecture Notes in Comput. Sci. 6410, Springer, Berlin, 2010, pp. 75–87.

12. P. Heggernes, D. Lokshtanov, R. Mihai, and C. Papadopoulos, *Cutwidth of split graphs, threshold graphs, and proper interval graphs,* SIAM J. Discrete Math., 25 (2011), pp. 1418–1437.
13. M. Jünger, G. Reinelt, and G. Rinaldi, *The traveling salesman problem,* In Handbook on Operations Research and Management Sciences, vol. 7, pp. 225–330, North-Holland, 1995.
14. D. R. Karger, *A randomized fully polynomial approximation scheme for the all terminal network reliability problem,* In Proceedings of the 27th Annual ACM Symposium on the Theory of Computing (STOC), ACM, 1995, pp. 11–17.
15. F. T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms,* In Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 1988, pp. 422–431.
16. F. Makedon and I. H. Sudborough, *Minimizing width in linear layouts,* In Proceedings of the 10th Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 154, Springer, Berlin, 1983, pp. 478–490.
17. B. Monien and I. H. Sudborough, *Min cut is NP-complete for edge weighted trees,* In Proceedings of the 10th Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 226, Springe,r Berlin, 1986, pp. 265–274.
18. P. Mutzel, *A polyhedral approach to planar augmentation and related problems,* In Proceedings of the 3rd Annual European Symposium on Algorithms (ESA), Lecture Notes in Comput. Sci. 979, Springer, Berlin, 1995, pp. 497–507.
19. J. Spinrad, A. Brandstädt, and L. Stewart, *Bipartite permutation graphs,* Discrete Appl. Math., 18 (1987), pp. 279–292.
20. D. M. Thilikos, M. J. Serna, and H. L. Bodlaender, *Cutwidth I: A linear time fixed parameter algorithm,* J. Algorithms, 56 (2005), pp. 1–24.
21. D. M. Thilikos, M. J. Serna, and H. L. Bodlaender, *Cutwidth II: Algorithms for partial w-trees of bounded degree,* J. Algorithms, 56 (2005), pp. 24–49.
22. M. Yannakakis, *A polynomial algorithm for the min cut linear arrangement of trees,* J. ACM, 32 (1985), pp. 950–988.
23. J. Yuan and S. Zhou, *Optimal labelling of unit interval graphs,* Appl. Math. J. Chinese Univ. Ser. B (English edition), 10 (1995), pp. 337–344.