Cutwidth of split graphs and threshold graphs*

Pinar Heggernes† Daniel Lokshtanov† Rodica Mihai† Charis Papadopoulos‡

Abstract

We give a linear-time algorithm to compute the cutwidth of threshold graphs, thereby resolving the computational complexity of cutwidth on this graph class. Threshold graphs are a well-studied subclass of interval graphs and of split graphs, both of which are unrelated subclasses of chordal graphs. To complement our result, we show that cutwidth is NP-complete on split graphs, and consequently also on chordal graphs. The cutwidth of interval graphs is still open, and only very few graph classes are known so far on which polynomial-time cutwidth algorithms exist. Thus we contribute to define the border between graph classes on which cutwidth is polynomially solvable and on which it remains NP-complete.

1 Introduction

The cutwidth problem asks, given a graph G, and a positive integer k, whether there exists a linear layout of the vertices of G so that any line inserted between two consecutive vertices of the layout cuts (intersects with) at most k edges. The cutwidth of the input graph is the smallest integer for which the question can be answered positively. This important graph layout problem was first proposed as a model to minimize the number of channels in a circuit [1, 19], and more recently it has found applications in areas like protein engineering [3], network reliability [15], automatic graph drawing [21], information retrieval [4], and as a subroutine in the cutting plane algorithm for TSP [14].

Like many other interesting graph problems, cutwidth is NP-complete [8], even when input graphs are restricted to planar graphs of maximum degree three [20], unit disk graphs, partial grids [9], and consequently bipartite graphs.

Coping with the NP-completeness of the problem has been mainly channelled via approximation algorithms and fixed parameter algorithms. There is a polynomial-time $O(\log^2 n)$-approximation algorithm for general graphs [17], and a polynomial-time constant factor approximation algorithm for dense graphs [2]. The best known parameterized algorithm for cutwidth so far runs in linear time (but of course exponential in the parameter k) [22].

Polynomial-time algorithms for the exact computation of cutwidth are known only for very few graph classes. For certain trivial graph classes, like meshes or complete p-partite graphs, there exist closed formulas for their cutwidth [10]. Cutwidth of proper interval graphs has a trivial solution following an interval ordering of the vertices [25]. The cutwidth of trees can

*This work is supported by the Research Council of Norway through grant 166429/V30. A preliminary version of this work was presented at WG 2008.

†Department of Informatics, University of Bergen, N-5020 Bergen, Norway. Emails: pinar@ii.uib.no, daniello@ii.uib.no, rodica@ii.uib.no

‡Department of Mathematics, University of Ioannina, GR-45110 Ioannina, Greece. Email: charis@cs.uoi.gr

be computed in $O(n \log n)$ time by a sophisticated and technical algorithm [24] (see also [6]). The cutwidth of graphs having bounded treewidth *and* maximum degree, can be computed in polynomial time by advanced methods [23]. The computational complexity of cutwidth on threshold graphs has been open until now [10].

In this paper, we present an $O(n)$-time algorithm for computing the cutwidth of threshold graphs with n vertices. Threshold graphs are a well-studied graph class with a variety of theoretical applications [18], and they are both split graphs and interval graphs [5, 12]. Split and interval graphs are two unrelated subclasses of the widely-known class of chordal graphs. Before presenting our algorithm for threshold graphs, we show that the cutwidth problem remains NP-complete on split graphs (even on a very restricted type of split graphs), and hence also on chordal graphs. Our findings are summarized in Figure 1.

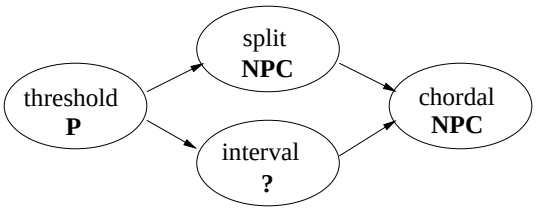


Figure 1: The graph classes studied in this paper, and the complexity of cutwidth on each class according to our results. P means polynomial and NPC means NP-complete. The arrow represents the subset relation.

The algorithm that we present for threshold graphs is simple and intuitive, and its execution does not at all depend on properties of threshold graphs; thus it can also be run on general graphs as a heuristic. Interestingly, while the algorithm correctly computes the cutwidth of threshold graphs, it does not compute the cutwidth of any known superclass or closely related class, like trivially-perfect graphs and chain graphs[1]. For the proof of correctness of this algorithm on threshold graphs, we study the properties of a possible minimal counterexample through a series of structural results, and we show that the assumption of the existence of such a counterexample leads to a contradiction.

2 Preliminaries

We consider labeled undirected finite graphs with no loops or multiple edges. For a graph $G = (V, E)$, we denote its vertex and edge set by V and E, respectively, with $n = |V|$. Every vertex $v \in V$ has a distinct label, label(v), between 1 and n. We say that a vertex u *is smaller than* v if label(u) $<$ label(v). For a vertex subset $S \subseteq V$, the subgraph of G induced by S is denoted by $G[S]$. Moreover, we denote by $G - S$ the graph $G[V \setminus S]$ and by $G - v$ the graph $G[V \setminus \{v\}]$. In this paper, we distinguish between *subgraphs* and *induced subgraphs*. By a *subgraph* of G we mean a graph G' on the same vertex set containing a subset of the edges of G, and we denote it by $G' \subseteq G$. If G' contains a proper subset of the edges of G, we write $G' \subset G$. We write $G - uv$ to denote the graph $(V, E \setminus \{uv\})$.

[1]For completeness, we give small examples illustrating this fact in an appendix at the end of the paper.

The *neighborhood* of a vertex x of G is $N_G(x) = \{v \mid xv \in E\}$. The *closed neighborhood* of x is $N_G[x] = N_G(x) \cup \{x\}$. The *degree* of x is $\Delta_G(x) = |N_G(x)|$. If $S \subseteq V$, then $N_G(S) = \bigcup_{x \in S} N_G(x) \setminus S$. We define the *cut* of S to be $\delta^G(S) = \{uv \in E \mid u \in S, v \notin S\}$, and the *cut size* of S to be $d^G(S) = |\delta^G(S)|$. Vertex x is *universal* if $N_G[x] = V$ and *isolated* if $N_G(x) = \emptyset$. We will omit the subscripts and superscripts when there is no ambiguity. A graph is *connected* if there is a path between any pair of vertices. A *connected component* of a disconnected graph is a maximal connected subgraph of it. A *clique* is a set of pairwise adjacent vertices, while an *independent set* is a set of pairwise non-adjacent vertices.

Given a graph $G = (V, E)$, a *layout* L is a one-to-one mapping $L : V \to \{1, \ldots, n\}$. We will also denote a layout L by $\langle v_1, v_2, \cdots v_n \rangle$ such that $L(v_i) = i$. For an integer i between 1 and n we define the set V_i to be $\{v_1, \cdots, v_i\}$. We say that u *is before* v in L, or $u <_L v$, if $L(u) < L(v)$. The *cut of* G *at the* i*th gap* in a given layout L is defined as $\delta_L(i) = \delta^G(V_i)$ and $d_L(i) = d^G(V_i)$. The *cutwidth* of a layout L of G is $\mathrm{cw}_L(G) = \max_{1 \le i \le n} d^G(V_i)$. The *cutwidth of* G is $\mathrm{cw}(G) = \min_L\{\mathrm{cw}_L(G)\}$ where the minimum is taken over all layouts of G. In this paper, an *optimal layout* of G is a layout L such that $\mathrm{cw}(G) = \mathrm{cw}_L(G)$.

The *bisection width* of G, denoted by $\mathrm{bw}(G)$, is the minimum cut size of any set $S \subset V$ on $\lfloor \frac{n}{2} \rfloor$ vertices. Since $\delta(S) = \delta(V \setminus S)$ it follows that $\mathrm{bw}(G)$ is the minimum cut size of S of any set S on $\lfloor \frac{n}{2} \rfloor$ or $\lceil \frac{n}{2} \rceil$ vertices. It should be clear that $\mathrm{bw}(G)$ gives a lower bound for $\mathrm{cw}(G)$, that is, $\mathrm{bw}(G) \le \mathrm{cw}(G)$ [10]. We will use the close connection between cutwidth and bisection width actively in some of our proofs. A useful observation is that the cutwidth of a subgraph G cannot exceed the cutwidth of G [10].

A graph is a *split graph* if its vertex set can be partitioned into a clique C and an independent set I, where (C, I) is called a *split partition*. A *threshold graph* is a split graph whose vertices can be ordered by neighborhood inclusion [12, 18]. Next we define a partitioning of the vertex set of a threshold graph that is used throughout the paper.

Definition 1. A *threshold partition* of a threshold graph $G = (V, E)$ is a partitioning of V into $(I_0 \ldots I_\ell, C_1 \ldots C_\ell)$, with $I = \bigcup_{i=0}^{\ell} I_i$ and $C = \bigcup_{i=1}^{\ell} C_i$ such that the following properties are satisfied.

- (C, I) is a split partition of G, that is, C is a clique and I is an independent set.
- Every vertex x in C has a neighbour in I.
- $N(I_1) \subset N(I_2) \subset \ldots \subset N(I_\ell)$ and for every integer $i \le \ell$ and vertices $u \in I_i$, $v \in I_i$, $N(u) = N(v)$.
- For every integer i between 1 and ℓ, $C_i = N(I_i) \setminus N(I_{i-1})$.

Every threshold graph has a threshold partition - we start with a split partition (C, I) of G with the largest cardinality of I among all split partitions of G. In such a partition, every vertex x of C has a neighbour in I, because otherwise $(C \setminus \{x\},\ I \cup \{x\})$ would also be a split partition of G with a larger I. Now, let $(I_0, I_1, I_2, \ldots, I_\ell)$ be the partitioning of I such that I_0 is the set of isolated vertices, and $N(I_1) \subset N(I_2) \subset \ldots \subset N(I_\ell)$, where ℓ is largest possible. For every integer i between 1 and ℓ define $C_i = N(I_i) \setminus N(I_{i-1})$. Now, $I_0 \ldots I_\ell, C_1 \ldots C_\ell$ is a threshold partition of G. We say that vertices of C_j and I_j belong to the jth *level* of the clique and of the independent set, respectively. By construction, the sets C_i and I_i are nonempty for every $1 \le i \le \ell$. For a vertex v, define level(v) to be the level that v belongs to. Notice that all vertices in I at the same level have the same degree, and that all vertices in C at the same level have the same degree.

Threshold graphs can be recognized, and their threshold partition can be computed, in $O(n+m)$ time [12]. Observe that a threshold partition of G completely defines G, since a vertex $u \in I$ and a vertex $v \in C$ are adjacent if and only if level$(u) \geq$ level(v). Our algorithm for computing the cutwidth of threshold graphs runs in $O(n)$ time if the threshold partition of G is given as input, and in $O(n+m)$ time if only an adjacency list representation is provided. We conclude this section with the following lemma.

Lemma 2.1 ([13]). *Let G be a threshold graph with threshold partition $(C_1, \ldots, C_\ell, I_1, \ldots, I_\ell)$. Let uv be an edge such that $u \in I_j$ and $v \in C_j$ for some j. Then $G - uv$ is a threshold graph.*

3 Cutwidth of split graphs

In this section we show that the cutwidth problem is NP-complete on split graphs. In fact the proof of Theorem 3.1 shows that cutwidth is NP-complete even on split graphs where every vertex of I in a split partition (C, I) has degree 2.

Theorem 3.1. *The cutwidth problem is NP-complete on split graphs.*

Proof. The reduction is from an arbitrary instance of the cutwidth problem. Given an arbitrary graph $G = (V, E)$ with n vertices and m edges, we construct a split graph G' as follows. To start with, G' is a complete graph on V. Let $k = n^2 + 1$. For every edge $uv \in E$ we add k more vertices to G', making each new vertex adjacent to u and v in G'. We say that these vertices of G' *correspond* to the edge uv of G. Observe that G' has $n + km$ vertices where the n vertices of V induce a clique in G'. The remaining km vertices are only adjacent to vertices of this clique. Hence G' is a split graph. Moreover the whole construction can be carried out in polynomial time. We now prove that for any $1 \leq c \leq n^2$ we have $\mathrm{cw}(G) \leq c$ if and only if $\mathrm{cw}(G') < c(k+1)$. (Note that n^2 is a trivial upper bound on the cutwidth of any graph on n vertices.)

If $\mathrm{cw}(G) \leq c$, then consider a layout L for which $\mathrm{cw}_L(G) \leq c$. We create a layout L' of G' by ordering the vertices in V in the same order that they have in L. Every vertex x of G' that corresponds to an edge uv of G is placed in an arbitrary position between u and v in L'. Observe that since x has degree 2 and is placed between its neighbors, $d_{L'}(L'(x)-1) = d_{L'}(L'(x))$, and thus, to compute $\mathrm{cw}_{L'}(G')$ it is sufficient to consider maximum $d_{L'}(L'(v))$ over all $v \in V$. From the construction of L' it follows that for every vertex v in V, $\delta_{L'}(L'(v))$ contains at most $k \cdot d_L(L(v))$ edges between vertices in V and vertices corresponding to edges of G, and at most n^2 edges between pairs of vertices in V. Thus

$$d_{L'}(L'(v)) \leq k \cdot d_L(L(v)) + n^2,$$

$$k \cdot d_L(L(v)) + n^2 < kc + k = k(c+1)$$

and $\mathrm{cw}(G') \leq \mathrm{cw}'_L(G') < k(c+1)$ follows.

Let L' be a layout of G' for which $\mathrm{cw}_{L'}(G') < k(c+1)$. ¿From L' we construct a layout L of G by ordering the vertices of V in the same order that L' orders them. We prove that $\mathrm{cw}_L(G) \leq c$. For a given vertex x we observe that for every edge $uv \in \delta_L(L(x))$ and vertex y of G' corresponding to the edge uv, either the edge yu or the edge yv must be in $\delta_{L'}(L'(x))$. Thus,

$$k \cdot d_L(L(x)) \leq d_{L'}(L'(x)) < k(c+1).$$

By dividing both sides by k we obtain $d_L(L(x)) < c+1$. Since we chose x arbitrarily, $\mathrm{cw}_L(G) \leq c$ and the result follows. □

4 Cutwidth of threshold graphs

In this section we give an algorithm that computes the cutwidth of threshold graphs in linear time. This algorithm constructs a layout $L = \langle v_1, v_2, \ldots, v_n \rangle$ by appending at step i the vertex v_i that minimizes the cut $\delta(V_{i-1} \cup \{v_i\})$.

For a given graph $G = (V, E)$ and a set $S \subseteq V$, we define the *rank* of a vertex v with respect to S to be $\text{rank}_S^G(v) = |N_G(v) \setminus S| - |N_G(v) \cap S|$ (superscript G is omitted when not needed). Observe that if $v \notin S$, then $d(S \cup \{v\}) = d(S) + \text{rank}_S(v)$. At step i, we select a vertex of $V \setminus V_{i-1}$ of lowest rank with respect to V_{i-1}. If there is a tie, the algorithm picks a vertex of highest degree. If there still is a tie, the algorithm picks the vertex with the smallest label between 1 and n distinctly assigned to each vertex prior to the algorithm. Note that this algorithm can be applied to arbitrary graphs. When G is a threshold graph with threshold partition (C, I), we assume that G has been labeled such that every vertex in I has smaller label than every vertex in C, for every pair u and v of vertices in I, $\text{level}(u) < \text{level}(v)$ implies $\text{label}(u) < \text{label}(v)$, and for every pair u and v of vertices in C, $\text{level}(u) < \text{level}(v)$ implies $\text{label}(u) < \text{label}(v)$. This can be easily achieved through an $O(n)$-time preprocessing step using the threshold partition of G.

The intuition behind the highest-degree tie-breaking is that when we add v to S, the ranks of all v's neighbors with respect to S decrease by 2, while the ranks of v's non-neighbors remain unchanged. Since we want the rank of the vertices we pick to be as small as possible, it is good to decrease the rank of as many vertices as possible. The details of the algorithm called MinCut are given below.

Algorithm: MinCut

Input: A graph $G = (V, E)$ with distinct labels between 1 and n on its vertices.
Output: A layout $L = \langle v_1, v_2, \cdots, v_n \rangle$ of G

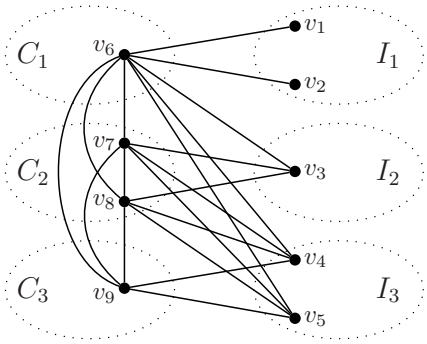
$V_0 := \emptyset$;
for i = 1 **to** n **do**
  $v_i :=$ the vertex in $V \setminus V_{i-1}$ with smallest label;
  for every vertex v in $V \setminus V_{i-1}$ ordered by increasing label **do**
    if $\text{rank}_{V_{i-1}}(v) < \text{rank}_{V_{i-1}}(v_i)$ **then** $v_i := v$
    else if $\text{rank}_{V_{i-1}}(v) = \text{rank}_{V_{i-1}}(v_i)$ and $\Delta(v) > \Delta(v_i)$ **then** $v_i := v$
  $V_i := V_{i-1} \cup \{v_i\}$;
  $L(v_i) := i$;

To illustrate the algorithm by an example, in Figure 2 (a) we depict a threshold graph on three levels with the corresponding label on each vertex. Execution of the Algorithm MinCut on this graph is given in Figure 2 (b) while the computed layout is shown in Figure 2 (c).

Before reaching the details of why Algorithm MinCut produces optimal layouts when the input is a threshold graph, we need to study how the layouts produced by the algorithm look. Observe first that if G has isolated vertices, then these can be placed in arbitrary positions in any optimal cutwidth layout, and our algorithm places them in the beginning of the output layout. For the statements of the following results in this section, we let $L = \langle v_1, \ldots, v_n \rangle$ be the layout computed by Algorithm MinCut when run on a threshold graph G with threshold partition $(C_1, \ldots, C_\ell, I_1, \ldots, I_\ell)$.

One should notice that in a threshold graph, two vertices with the same degree have the same open neighborhood if they are nonadjacent and the same closed neighborhood if they are adjacent. Thus, in threshold graphs, the labels of the vertices do not really affect the layout

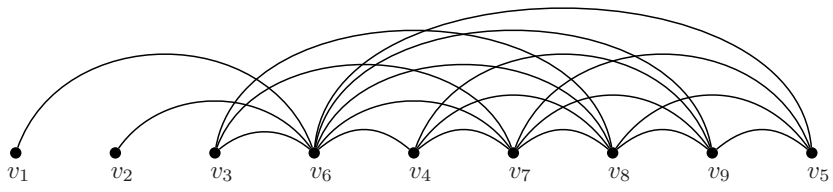
produced by Algorithm MinCut, because whether or not there is an edge between v_i and v_j for $1 \leq i < j \leq n$ is independent of the labels of the vertices, and depends only on which sets of the threshold partition they belong to. The reason we include the labels in the description of the algorithm is that the labels simplify the discussions in the proofs. One should note that the labels do indeed affect the layout produced by the algorithm if the algorithm is run on a graph



(a)

step	rank									selection
	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	
1	(1)	1	3	4	4	8	6	6	5	v_1
2		(1)	3	4	4	6	6	6	5	v_2
3			(3)	4	4	4	6	6	5	v_3
4				4	4	(2)	4	4	5	v_6
5				(2)	2		2	2	3	v_4
6					2		(0)	0	1	v_7
7					0			(-2)	-1	v_8
8					-2				(-3)	v_9
9					(-4)					v_5

(b)



(c)

Figure 2: (a) A threshold graph on three levels where each vertex is labeled according to a preprocessing step. (b) Steps of the Algorithm MinCut applied on the threshold graph. (c) A layout of cutwidth 9 produced by the algorithm.

that is not a threshold graph.

Given two vertices u and v of G such that $u \in I$ and $v \in C$, we define the vertex set $\text{over}(u, v)$ to contain all vertices $x \in I$ such that $\text{label}(x) < \text{label}(u)$ and all vertices $y \in C$ such that $\text{label}(y) < \text{label}(v)$. The essence of the following lemma is that the algorithm picks vertices at lower levels before proceeding to higher levels, and that it starts with a vertex of I.

Lemma 4.1. *For any $i \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, \ell\}$:*

(a) If $V_i \cap I_k \neq \emptyset$, then $I_{k'} \subseteq V_i$, for every $1 \leq k' < k$.

(b) If $V_i \cap C_k \neq \emptyset$, then $C_{k'} \subseteq V_i$, for every $1 \leq k' < k$.

(c) If $V_i \cap I_k = \emptyset$, then $V_i \cap C_k = \emptyset$.

Proof. We prove all the statements simultaneously by induction on i. Vertex v_1 is of minimum degree and hence $v_1 \in I_1$, so for $i = 1$ the statements are trivially true. Assume now that all three statements hold whenever $i < r$ and consider the r'th step of the algorithm. We first prove that (a) must be true for $i = r$. It suffices to show that if $v_r \in I$, then v_r is a vertex with the lowest degree out of the ones that are not in V_{r-1}. Indeed, consider two vertices u and $v \in I \setminus V_{r-1}$ such that $\Delta(u) < \Delta(v)$. Since $\text{level}(v) > \text{level}(u)$, because (c) holds for V_{r-1} by the induction hypothesis, every neighbor of v that is not a neighbor of u is not in V_{r-1}. Thus $\text{rank}_{V_{r-1}}(u) < \text{rank}_{V_{r-1}}(v)$ and (a) follows for $i = r$.

We prove that (b) is true for $i = r$; that is, if $v_r \in C$, then v_r is a vertex with the highest degree out of the ones that are not in V_{r-1}. Let t be the largest integer such that $V_{r-1} \cap I_t \neq \emptyset$. For a vertex $u \in C_{t'}$ with $t < t'$, let v be a vertex in $I_{t'}$. Because (c) holds for V_{r-1} by the induction hypothesis, $v \notin V_{r-1}$ and $\text{rank}_{V_{r-1}}(v) \leq \text{rank}_{V_{r-1}}(u)$. Additionally, $\text{label}(v) < \text{label}(u)$ and so the algorithm would not pick v_r to be u. Furthermore, for every two vertices u and v in $C \setminus V_{r-1}$ such that $\text{level}(u) < \text{level}(v) \leq t$, it follows that every neighbor of u that is a non-neighbor of v is in V_{r-1}, yielding $\text{rank}_{V_{r-1}}(u) < \text{rank}_{V_{r-1}}(v)$ and completing the proof of (b) for $i = r$.

Finally, we prove that (c) is true for $i = r$. Consider a level t such that $I_t \cap V_{r-1} = C_t \cap V_{r-1}$ and let $u \in I_t$ and $v \in C_t$. Since (a) and (b) are true for $r - 1$, every neighbor of v that is not a neighbor of u is not in V_{r-1}. Unless $t = \ell$ and $|I_\ell| = 1$, $\Delta(u) < \Delta(v)$ so $\text{rank}_{V_{r-1}}(u) < \text{rank}_{V_{r-1}}(v)$. If $t = \ell$ and $|I_\ell| = 1$, then u and v have the same closed neighborhood, but $\text{label}(u) < \text{label}(v)$. In both cases $u <_L v$, and so (c) must be true for $i = r$. □

As a direct consequence of Lemma 4.1 (a) and (b), for any i between 1 and n, if $u \in I \cap V_i$ and $v \in C \cap V_i$, then $\text{over}(u, v) \subseteq V_i$.

Lemma 4.2. *Let $u \in I$ and $v \in C$. Then $u <_L v$ if and only if u and v are non-adjacent or $\text{rank}_{\text{over}(u,v)}(u) < \text{rank}_{\text{over}(u,v)}(v)$.*

Proof. We prove the lemma by showing *(i)*; if u and v are non-adjacent, then $u <_L v$, and *(ii)*; if u and v are adjacent then $u <_L v$ if and only if $\text{rank}_{\text{over}(u,v)}(u) < \text{rank}_{\text{over}(u,v)}(v)$. Let u and v be non-adjacent. Then, clearly $\text{level}(u) < \text{level}(v)$ and $u <_L v$ by Lemma 4.1, proving *(i)*.

Now, suppose u and v are adjacent and $u <_L v$. We prove that $\text{rank}_{\text{over}(u,v)}(u) < \text{rank}_{\text{over}(u,v)}(v)$. Clearly, $\text{rank}_{V_{L(u)-1}}(u) < \text{rank}_{V_{L(u)-1}}(v)$. Furthermore, every vertex in $\text{over}(u, v) \setminus V_{L(u)-1}$ is adjacent to both u and v. Thus $\text{rank}_{\text{over}(u,v)}(u) < \text{rank}_{\text{over}(u,v)}(v)$. Similarly, if u and v are adjacent and $v <_L u$, then $\text{rank}_{V_{L(v)-1}}(u) \geq \text{rank}_{V_{L(v)-1}}(v)$. Now, every vertex in $\text{over}(u, v) \setminus V_{L(v)-1}$ is nonadjacent to u. Hence $\text{rank}_{\text{over}(u,v)}(u) \geq \text{rank}_{\text{over}(u,v)}(v)$, proving *(ii)*. □

We say that the algorithm *covers* a vertex set S if there is an index i such that $V_i = S$. If the algorithm covers S, then we also say that S is covered.

Lemma 4.3. *Let $u, u' \in I$ with label(u') = label$(u) + 1$, and let $v, v' \in C$ with label(v') = label$(v) + 1$. Then over(u', v') is covered if and only if $u <_L v'$ and $v <_L u'$.*

Proof. If over(u', v') is covered, then over$(u', v') = V_i$ with $L(u) \leq i$, $L(v) \leq i$, $L(u') > i$ and $L(v') > i$ so $u <_L v'$ and $v <_L u'$. In the other direction, suppose $u <_L v'$ and $v <_L u'$. Let $S' = V_i$ be the smallest covered set that contains both u and v. By Lemma 4.1, over$(u', v') = \{u, v\} \cup \text{over}(u, v) \subseteq S'$. Furthermore, since S' is the smallest covered set that contains both u and v, either $v_i = u$ or $v_i = v$. If $v_i = u$, then $v_i <_L u'$ by Lemma 4.1 and $v_i <_L v'$ by assumption, while if $v_i = v$, then $v_i <_L u'$ by assumption and $v_i <_L v'$ by Lemma 4.1. In both cases, neither u' nor v' can be in S' which means that $S = S'$ and that S is covered. □

We are now equipped with most of the tools that are necessary to work with layouts produced by Algorithm MinCut. All that remains before we move on to proving the correctness of the algorithm are a couple of simple observations.

Observation 4.4. *For each integer $i \leq n - 1$, rank$_{V_i}(v_{i+1}) \geq$ rank$_{V_{i-1}}(v_i) - 2$. Furthermore, if $\Delta(v_{i+1}) > \Delta(v_i)$, then rank$_{V_i}(v_{i+1}) \geq$ rank$_{V_{i-1}}(v_i) - 1$.*

Proof. Observe that if v_{i+1} and v_i are adjacent, then $\text{rank}_{V_i}(v_{i+1}) = \text{rank}_{V_{i-1}}(v_{i+1}) - 2$; otherwise, $\text{rank}_{V_i}(v_{i+1}) = \text{rank}_{V_{i-1}}(v_{i+1})$. Since the algorithm picked v_i and not v_{i+1} at step i, the observation follows. □

Observation 4.5. *For every level $k \leq \ell$ and every triple of vertices $u, v \in C_k$ and $w \notin C_k$, $u <_L w$ if and only if $v <_L w$.*

Proof. If $w \in C$ the observation follows from Lemma 4.1 (b). If $w \in I$ and $uw \notin E$, then $vw \notin E$ and the observation follows from Lemma 4.2. If $w \in I$ and $uw \in E$, then $vw \in E$. Without loss of generality, label$(u) <$ label(v). Note that every vertex in over$(w, v) \setminus$ over(w, u) is in C_k and so $\text{rank}_{\text{over}(w,u)}(w) < \text{rank}_{\text{over}(w,u)}(u)$ if and only if $\text{rank}_{\text{over}(w,v)}(w) < \text{rank}_{\text{over}(w,v)}(v)$. Applying Lemma 4.2 completes the proof. □

4.1 Correctness of Algorithm MinCut

In this subsection, we show that Algorithm MinCut produces optimal layouts when the input is a threshold graph. We assume for contradiction that there is a threshold graph $G = (V, E)$ with threshold partition $(C_1, \ldots, C_\ell, I_1, \ldots, I_\ell)$ on which Algorithm MinCut outputs layout $L = \langle v_1, \ldots, v_n \rangle$ such that $\text{cw}_L(G) > \text{cw}(G)$. We call such a threshold graph a *counterexample*, and we say that a counterexample is *minimal* if it is has the smallest value of $|V| + |E|$ among all counterexamples. A *bad set* of counterexample G is a set $S \subseteq V$ that is covered by the algorithm and for which $d(S) > \text{cw}(G)$. A *locally worst* bad set is a bad set $S = V_i$ such that $d(S) \geq d(V_{i+1})$ and $d(S) \geq d(V_{i-1})$. Observe that $\text{rank}_{V_{i-1}}(v_i)$ must be non-negative and $\text{rank}_{V_i}(v_{i+1})$ must be non-positive for V_i to be locally worst. Observation 4.4 then implies that $\text{rank}_{V_{i-1}}(v_i)$ is 2, 1 or 0. This means that if $v_i \in C_1$, then i is $\lfloor \frac{n}{2} \rfloor$, $\frac{n}{2}$ or $\lceil \frac{n}{2} \rceil$ respectively. Another thing to notice about locally worst bad sets is that if both V_i and V_{i-1} are bad sets with $d(V_{i-1}) \geq d(V_i)$, then some locally worst bad set is a strict subset of V_i.

The main idea of the proof is to show that if there is a counterexample G, then there must be another counterexample G' that either has at most 2 levels, or exactly 3 levels and a very specific structure. We complement this result by showing that the algorithm produces optimal layouts on all graphs with at most 2 levels, and on all graphs with 3 levels and the mentioned structural properties. This yields that $\mathrm{cw}_L(G) = \mathrm{cw}(G)$ for every threshold graph G.

Lemma 4.6. *Let G be a threshold graph on exactly 1 level. Then $cw_L(G) = cw(G)$.*

Proof. Let (C, I) be a threshold partition of G. Observe that every vertex of I is adjacent to every vertex of C. The algorithm lays out $\lfloor \frac{|I|}{2} \rfloor$ vertices of I, then all of C, followed by the remaining vertices of I. By inspection,

$$\mathrm{cw}_L(G) = \left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lceil \frac{n}{2} \right\rceil - \left\lfloor \frac{|I|}{2} \right\rfloor \cdot \left\lceil \frac{|I|}{2} \right\rceil.$$

Since all non-edges of G are between vertices in I,

$$\mathrm{bw}(G) = \left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lceil \frac{n}{2} \right\rceil - \left\lfloor \frac{|I|}{2} \right\rfloor \cdot \left\lceil \frac{|I|}{2} \right\rceil.$$

Thus $\mathrm{cw}_L(G) = \mathrm{bw}(G) \leq \mathrm{cw}(G) \leq \mathrm{cw}_L(G)$ and $\mathrm{cw}_L(G) = \mathrm{cw}(G)$ follows. □

Already for threshold graphs with 2 levels, the correctness proof for Algorithm MinCut is more complicated. Before we go on to this proof we need more tools to work with locally worst bad sets.

For the statements of all the remaining results and definitions in this section, whenever we mention a counterexample G, we let $(C_1, \ldots, C_\ell, I_1, \ldots, I_\ell)$ be its threshold partition. Recall that the output of Algorithm MinCut is always denoted by $L = \langle v_1, \ldots, v_n \rangle$.

Lemma 4.7. *Every locally worst bad set S of a counterexample G satisfies*

(i) $C_1 \cap S \neq \emptyset$, *(ii)* $I_1 \subseteq S$, *(iii)* $C_\ell \cap S = \emptyset$, *and* *(iv)* $I_\ell \setminus S \neq \emptyset$.

Proof. (i) If $S \cap C_1 = \emptyset$ and $v_{i+1} \in I$, then $\mathrm{rank}_{V_i}(v_{i+1}) = \Delta(v_{i+1}) > 0$ contradicting that S is locally worst. If $S \cap C_1 = \emptyset$, $v_{i+1} \in C$ and $\mathrm{rank}_{V_i}(v_{i+1}) > 0$, then S is not locally worst, so $\mathrm{rank}_{V_i}(v_{i+1}) \leq 0$. Since $v_i \in I$ and $\mathrm{rank}_{V_{i-1}}(v_i) = \Delta(v_i) > 0$, Observation 4.4 implies that $\mathrm{rank}_{V_i}(v_{i+1}) = 0$ and that $\mathrm{rank}_{V_{i-1}}(v_i) = 1 = \Delta(v_i)$. This means n is odd, and $i = \lfloor \frac{n}{2} \rfloor$. Hence $d(S) = \lfloor \frac{n}{2} \rfloor$ and since v_{i+1} is a universal vertex of G, $\lfloor \frac{n}{2} \rfloor \leq \mathrm{bw}(G)$. Thus $d(S) = \lfloor \frac{n}{2} \rfloor \leq \mathrm{bw}(G) \leq \mathrm{cw}(G)$ contradicting that S is a bad set. We can conclude that $S \cap C_1 \neq \emptyset$.

(ii) Suppose for contradiction that $I_1 \setminus S \neq \emptyset$. By Lemma 4.1, $S \subset I_1 \cup C_1$. By the discussion in the previous paragraph, $S \cap C_1 \neq \emptyset$. If $v_i \in I_1$, then Observation 4.5 implies $C_1 \subseteq V_{i-1}$ so $\mathrm{rank}_{V_{i-1}}(v_i) = -\Delta(v_i) < 0$, contradicting that S is locally worst. Thus $v_i \in C_1$.

Let I_f be the subset of I_1 that L puts before C_1, and C_f be the $\lceil \frac{|C_1|}{2} \rceil$ vertices of C_1 with smallest labels. Observation 4.5 guarantees that the set $S' = I_f \cup C_f$ is covered, so $S' = V_j$. We prove that S' is a bad set. Let x be the vertex of C_1 with the smallest label and let $k = L(x)$. Since the algorithm chose x over a vertex in I_1 in step k, $\mathrm{rank}_{V_{k-1}}(x) \leq |C_1|$. By Observation 4.4, $\mathrm{rank}_{V_{k-1}}(x) \geq |C_1| - 1$. Now, for every $k \leq t < k + |C_1|$ we have $\mathrm{rank}_{V_{t-1}}(v_t) =$

$\text{rank}_{V_{k-1}}(x) - 2(t-k)$. Note that since $v_i \in C_1$, $i < k + |C_1|$. Since $|C_f| = \lceil \frac{|C_1|}{2} \rceil$, we have $1 + (j-k) = \lceil \frac{|C_1|}{2} \rceil$. Thus

$$\text{rank}_{V_{j-1}}(v_j) = \text{rank}_{V_{k-1}}(x) - 2(j-k) = \text{rank}_{V_{k-1}}(x) - 2(\left\lceil \frac{|C_1|}{2} \right\rceil - 1).$$

This in turn implies

$$|C_1| + 1 - 2\left\lceil \frac{|C_1|}{2} \right\rceil \leq \text{rank}_{V_{j-1}}(v_j) \leq |C_1| + 2 - 2\left\lceil \frac{|C_1|}{2} \right\rceil.$$

Simplifying the bounds yields $0 \leq \text{rank}_{V_{j-1}}(v_j) \leq 2$. Therefore, for every t with $k \leq t < j$, $\text{rank}_{V_{t-1}}(v_t) \geq 0$, while for every t such that $j < t \leq i$, $\text{rank}_{V_{t-1}}(v_t) \leq 0$. Thus $d(S') \geq d(S)$.

Let G' be the graph obtained from G by removing all edges that do not have at least one endpoint in C_1. The threshold partition of G' is $(C_1, V \setminus C_1)$ where every vertex of C_1 is universal. S' contains $\lceil \frac{|C_1|}{2} \rceil$ vertices of C_1 and exactly $\lfloor \frac{n-|C_1|}{2} \rfloor$ vertices of the independent set of the threshold partition of G'. Furthermore, none of the removed edges have one endpoint in S' and one outside of S', hence $d^{G'}(S') = d(S')$. Now, since all non-edges of G' are between vertices in $V \setminus C_1$ we have that

$$d^{G'}(S') = \left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lceil \frac{n}{2} \right\rceil - \left\lfloor \frac{n-|C_1|}{2} \right\rfloor \cdot \left\lceil \frac{n-|C_1|}{2} \right\rceil = \text{bw}(G').$$

Thus $d(S) \leq d(S') = d^{G'}(S') = \text{bw}(G') \leq \text{cw}(G') \leq \text{cw}(G)$ contradicting that S is a bad set. We can conclude that $I_1 \subseteq S$.

(iii) Suppose for contradiction that $S \cap C_\ell \neq \emptyset$. By Lemma 4.6 G has at least two levels. If $|I_\ell| = 1$, then all edges of $\delta(S)$ are between vertices of $C \cup I_\ell$ which induces a clique in G. In that case $d(S) \leq \text{cw}(G[C \cup I_\ell]) \leq \text{cw}(G)$ contradicting that S is a bad set. Thus, $|I_\ell| > 2$. If $v_i \in I_\ell$, Lemma 4.5 implies that $C_\ell \subseteq V_{i-1}$ which in turn implies $C \subseteq V_{i-1}$. Then $\text{rank}_{V_{i-1}}(v_i) = -\Delta(v_i) < 0$, contradicting that S is locally worst. Let I_f be the set of the $\lfloor \frac{|I_\ell|}{2} \rfloor$ vertices of I_ℓ with smallest labels. Let u be the vertex in I_f with the largest label, u' be the vertex in I_ℓ with $\text{label}(u') = \text{label}(u) + 1$ and v be the vertex in C_ℓ with the smallest label.

We wish to show that $u <_L v <_L u'$. Notice that v is the only neighbor of u and u' that is a non-neighbor of v and that $v \notin \text{over}(u, v)$ and $v \notin \text{over}(u', v)$. Among the neighbors of v, apart from u, that are non-neighbors of u, there are

$$\left\lfloor \frac{|I_\ell|}{2} \right\rfloor - 1 \text{ in over}(u, v) \text{ and } I_\ell - \left\lfloor \frac{|I_\ell|}{2} \right\rfloor \geq \left\lfloor \frac{|I_\ell|}{2} \right\rfloor \text{ outside.}$$

Thus by Lemma 4.2, $u <_L v$. Similarly, among the neighbors of v, apart from u', that are non-neighbors of u', there are

$$\left\lfloor \frac{|I_\ell|}{2} \right\rfloor \text{ in over}(u, v) \text{ and } I_\ell - \left\lfloor \frac{|I_\ell|}{2} \right\rfloor - 1 \leq \left\lfloor \frac{|I_\ell|}{2} \right\rfloor \text{ outside.}$$

Since $\Delta(u') < \Delta(v)$ by Lemma 4.2, $v <_L u'$. Thus $S \cap I_\ell = I_f$.

Notice that all the edges of $\delta(S)$ are between vertices in $I_\ell \cup C$. Thus, if we let $S' = I_f \cup (C \cap S)$, we have $d(S) = d^{G[I_\ell \cup C]}(S')$. Now $G[I_\ell \cup C]$ has threshold partition (C, I_ℓ) and every vertex of C is adjacent to every vertex of I_ℓ. S' contains exactly $\lfloor \frac{|I_\ell|}{2} \rfloor$ vertices of I_ℓ and thus

$$d^{G[I_\ell \cup C]}(S') \leq \left\lfloor \frac{|I_\ell \cup C|}{2} \right\rfloor \cdot \left\lceil \frac{|I_\ell \cup C|}{2} \right\rceil - \left\lfloor \frac{|I_\ell|}{2} \right\rfloor \cdot \left\lceil \frac{|I_\ell|}{2} \right\rceil = \text{bw}(G[I_\ell \cup C]).$$

Hence $d(S) \leq d^{G[I_\ell \cup C]}(S') \leq \text{bw}(G[I_\ell \cup C]) \leq \text{cw}(G[I_\ell \cup C]) \leq \text{cw}(G)$ contradicting that S is a bad set. Thus $S \cap C_\ell = \emptyset$.

(iv) Suppose for contradiction that $I_\ell \subseteq S$. If $|I_\ell| = 1$, let u be the vertex with the smallest label in I_ℓ and let v be the vertex with the largest label in $C_{\ell-1}$. Every neighbor of v apart from u that is a non-neighbor of u lies in $\text{over}(u, v)$, thus by Lemma 4.2, $v <_L u$ and $C_1 \subset S$. Then every edge of $\delta(S)$ is between vertices of $C \cup I_\ell$. Since $C \cup I_\ell$ induces a clique in G this contradicts that S is a bad set. Thus $|I_2| > 1$.

If $|I_2| > 1$ let u be the vertex with the highest label in I_ℓ and v be the vertex with the highest label in C_ℓ. Every vertex that is neighbor of v apart from u that is a non-neighbor of u lies in $\text{over}(u, v)$ thus, by Lemma 4.2 $v <_L u$ and $C_\ell \subset S$. This contradicts that any locally worst bad set S of a counterexample G contains no vertices of C_ℓ. □

Lemma 4.8. *Let G be a threshold graph on exactly 2 levels. Then $cw_L(G) = cw(G)$.*

Proof. Suppose for contradiction that G is a counterexample on two levels with minimum value of $|V|+|E|$ among all counterexamples on two levels. Lemma 4.6 ensures that G is a counterexample with minimum value of $|V| + |E|$ among all counterexamples on *at most* two levels, but G is not necessarily a minimal counterexample. Since G is a counterexample, G has a locally worst bad set $S = V_i$. Let S be the smallest locally worst bad set of G. We distinguish between two cases:

$$(i)\ |C_1| < |C_2| \quad \text{and} \quad (ii)\ |C_1| \geq |C_2|.$$

(i) We first consider the case that $|C_1| < |C_2|$ and $S \cap I_2 = \emptyset$. By Lemma 4.1, $S \cap C_2 = \emptyset$. By Lemma 4.7, $S \cap C_1 \neq \emptyset$. If $v_i \in I_1$, then Observation 4.5 implies $C_1 \subseteq V_{i-1}$, so $\text{rank}_{V_{i-1}}(v_i) = -\Delta(v_i) < 0$, contradicting that S is locally worst. Thus $v_i \in C_1$. Furthermore, by Lemma 4.7, $I_1 \subseteq S$. If $v_{i+1} \in I_2$, by assumption, we have $|C_1| < |C_2|$. In this case $\text{rank}_{V_i}(v_{i+1}) > 0$, contradicting that S is locally worst. Therefore v_{i+1} must be in C_1.

Let u be the vertex of I_1 with highest label, u' be the vertex of I_2 with smallest label, $v = v_i$ and $v' = v_{i+1}$. Observe that $S = \text{over}(u', v')$. Let x be the vertex in C_2 with highest label. By Lemma 4.2, $v <_L u'$ if and only if $\text{rank}_{\text{over}(u',v)}(u') \geq \text{rank}_{\text{over}(u',v)}(v)$. Now

$$\text{rank}_{\text{over}(u',v)}(u') - \text{rank}_{\text{over}(u',v)}(v) = 1 + |I_1| - |I_2|.$$

Suppose $1 + |I_1| > |I_2|$. Then $\text{rank}_{\text{over}(u',v)}(u') > \text{rank}_{\text{over}(u',v)}(v)$. We make a new graph G' from G by deleting all edges between x and vertices in I_2 and relabeling the vertices such that the labeling order of all vertices is the same as in G, with the exception of x which gets the highest label among the vertices of the independent set of the threshold partition of G'. Observe that G' is a threshold graph on at most 2 levels, and that G' is a proper subgraph of G. In addition $\text{rank}_{\text{over}(u,v')}(u)$, $\text{rank}_{\text{over}(u,v')}(v')$ and $\text{rank}_{\text{over}(u',v)}(v)$ remain unchanged from G to G' while $\text{rank}^{G'}_{\text{over}(u',v)}(u') = \text{rank}^{G}_{\text{over}(u',v)}(u') - 1$. In G we had

$$\text{rank}_{\text{over}(u,v')}(u) < \text{rank}_{\text{over}(u,v')}(v') \quad \text{and} \quad \text{rank}_{\text{over}(u',v)}(u') > \text{rank}_{\text{over}(u',v)}(v).$$

Hence in G' we know that

$$\text{rank}^{G'}_{\text{over}(u,v')}(u) < \text{rank}^{G'}_{\text{over}(u,v')}(v') \quad \text{and} \quad \text{rank}^{G'}_{\text{over}(u',v)}(u') \geq \text{rank}^{G'}_{\text{over}(u',v)}(v).$$

Lemma 4.2 yields $u <_{L'} v'$ and $v <_{L'} u'$. By Lemma 4.3, S is covered also by L'. Since we did not delete any edges in $\delta(S)$ it follows that S is a bad set also in G'. This contradicts that G is a counterexample on at most 2 levels with the smallest value of $|V| + |E|$.

To conclude the case where $|C_1| < |C_2|$ and $S \cap I_2 = \emptyset$, suppose that $1 + |I_1| = |I_2|$. Then $S \subseteq I_1 \cup C_1$, $|I_1| < |I_2|$ and $|C_1| < |C_2|$ together imply that $|S| \leq |V \setminus S| + 2$. But $\text{rank}_{V_{i-1}}(v_i) \leq 2$ implies $|S| \geq \lfloor \frac{n}{2} \rfloor$, which is a contradiction.

Now, consider the case where $|C_1| < |C_2|$ and $S \cap I_2 \neq \emptyset$. If $|I_2| = 1$, then $I_2 \subseteq S$, contradicting Lemma 4.7. Thus $|I_2| > 1$. Suppose $C_1 \setminus S \neq \emptyset$. Let u be the vertex in $I_2 \cap S$ with highest label and x be the vertex in C_1 with lowest label. By Lemma 4.7, $x \in S$. Since $C_1 \setminus S \neq \emptyset$, Observation 4.5 implies that $u <_L x$. Since u is the vertex in $I \cap S$ with highest label and x be the vertex in $C \cap S$ with lowest label, $L(x) = L(u) + 1$ and $v_{i+1} \in C_1$. By Observation 4.4, $\text{rank}_{V_{L(x)-1}}(x) \geq |C| - 1$. Thus

$$\text{rank}_{V_i}(v_{i+1}) \geq |C| - 1 - 2(|C_1| - 1) = |C_2| + 1 - |C_1| > 0,$$

contradicting that S is locally worst. Thus $C_1 \subseteq S$.

If $v_{i+1} \in I_2$, then by Lemma 4.7, $\text{rank}_{V_i}(v_{i+1}) \geq |C_2| - |C_1| > 0$ contradicting that S is locally worst. Thus $v_{i+1} \in C_2$. Furthermore as $S \cap C_2 = \emptyset$ this means that v_{i+1} is the vertex of C_2 with lowest rank and that $|S \cap I_2| = \lfloor \frac{I_2}{2} \rfloor$. Thus,

$$d(S) \leq d^{G[I_2 \cup C]}((S \cap I_2) \cup (S \cap C)) \leq \text{cw}(G[I_2 \cup C]) \leq \text{cw}(G),$$

contradicting that S is a bad set. This concludes the case that $|C_1| < |C_2|$.

(ii) If $|C_1| \geq |C_2|$, then $v_i \in C_1$ because otherwise $C_1 \subseteq V_{i-1}$ and $\text{rank}_{V_{i-1}}(v_i) \leq |C_2| - |C_1| \leq 0$ contradicting that S is the *smallest* locally worst bad set. Let u be the vertex in I_1 with the highest label, and let $v = v_i$. Lemma 4.7 implies $I_1 \subseteq S$ and hence $u <_L v$. By applying Lemma 4.2 on u and v we conclude that $\text{rank}_{\text{over}(u,v)}(v) - \text{rank}_{\text{over}(u,v)}(u) = |I_2| + |C_2| - (|I_1| - 1) > 0$, so $|I_1| \leq |I_2| + |C_2|$. Furthermore, since S is the smallest locally worst bad set, $\text{rank}_{V_{i-1}}(v)$ is either 1 or 2, so $|S| = \lfloor \frac{n}{2} \rfloor$.

Now, let S' be a set of vertices in $\overline{G}$ maximizing $d^{\overline{G}}(S')$. Notice that

$$\text{bw}(G) \geq \left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lceil \frac{n}{2} \right\rceil - d^{\overline{G}}(S').$$

We will use this fact to show that S can not be a bad set and obtain a contradiction. Since $\delta^{\overline{G}}(S') = \delta^{\overline{G}}(V \setminus S')$, without loss of generality we assume $|I_1 \cap S'| \geq |I_1 \setminus S'|$. In $\overline{G}$ all the neighbours of a vertex in C_2 are in I_1. Thus every vertex in C_2 has at least as many neighbours as non-neighbours in S' in $\overline{G}$. Hence $d^{\overline{G}}(S' \setminus C_2) \geq d^{\overline{G}}(S')$ and so we assume that $C_2 \cap S' = \emptyset$. Also, if $x \in I_1 \setminus S'$ and $y \in I_2 \cap S'$, observe that $d^{\overline{G}}(S' \cup \{x\} \setminus \{y\}) > d^{\overline{G}}(S')$, contradicting the choice of S'. Therefore, if $I_2 \cap S' \neq \emptyset$, then $I_1 \subseteq S'$. Suppose now that $x \in I_1 \setminus S'$. Then $I_2 \cap S' = \emptyset$ and since $|I_1| \leq |I_2| + |C_2|$, $d^{\overline{G}}(S' \cup \{x\}) > d^{\overline{G}}(S')$, contradicting the choice of S'. Therefore we conclude that $I_1 \subseteq S'$ and $C_2 \cap S' = \emptyset$.

If $|I_2| \leq |I_1| + 1$, then every vertex x in I_2 has at least as many neighbours in I_1 as in I_2 in $\overline{G}$. Hence $d^{\overline{G}}(S' \setminus \{x\}) \geq d^{\overline{G}}(S)$ and so without loss of generality $S' \cap I_2 = \emptyset$ and $S' = I_1$. Let x be the vertex of I_2 with lowest label and y be the vertex of C_1 with highest label. Now, $\text{rank}_{\text{over}(x,y)}(y) - \text{rank}_{\text{over}(x,y)}(x) = |I_2| - 1 - |I_1| \leq 0$ and hence by Lemma 4.2, $y <_L x$. Since $v_i \in C_1$ it follows that $S \subseteq I_1 \cup C_1$. Recall that by Lemma 4.7, $I_1 \subseteq S$. Since vertices in C_1 are

isolated in $\overline{G}$ it follows that $d^{\overline{G}}(S) = d^{\overline{G}}(S')$, so $\text{bw}(G) \geq \lfloor \frac{n}{2} \rfloor \cdot \lceil \frac{n}{2} \rceil - d^{\overline{G}}(S)$. In addition, since $|S| = \lfloor \frac{n}{2} \rfloor$ we have $\lfloor \frac{n}{2} \rfloor \cdot \lceil \frac{n}{2} \rceil - d^{\overline{G}}(S) = d(S)$. This implies $\text{bw}(G) \geq d(S)$, contradicting that S is a bad set.

If $|I_2| > |I_1| + 1$, by Lemma 4.2 applied to the vertex of I_2 with lowest label and the vertex of C_1 with highest label, S contains at least one vertex of I_2. By Lemma 4.7 S does not contain all of I_2 so we can apply Lemma 4.2 again to obtain that $|S \cap I| = \lfloor \frac{|I|}{2} \rfloor$. Furthermore, if $|S' \cap I| > \lfloor \frac{|I|}{2} \rfloor$, let $y \in S' \cap I$. Then $d^{\overline{G}}(S' \setminus \{y\}) \geq d^{\overline{G}}(S')$, so without loss of generality $|S' \cap I| \leq \lfloor \frac{|I|}{2} \rfloor$. Similarly, if $|S' \cap I| < \lfloor \frac{|I|}{2} \rfloor$, let $y \in I \setminus S'$. Then $d^{\overline{G}}(S' \cup \{y\}) \geq d^{\overline{G}}(S')$, so without loss of generality $|S' \cap I| \geq \lfloor \frac{|I|}{2} \rfloor$. Thus, without loss of generality $|S' \cap I| = \lfloor \frac{|I|}{2} \rfloor$. Since vertices in C_1 are isolated in $\overline{G}$ it follows that $d^{\overline{G}}(S) = d^{\overline{G}}(S')$. As in the previous paragraph, this implies $\text{bw}(G) \geq d(S)$, contradicting that S is a bad set. □

From the above lemma it follows that any counterexample has at least 3 levels. Over the next few lemmas, we show how any counterexample can be transformed into a counterexample with exactly 3 levels. The first observation is that in every minimal counterexample all parts of the graph participate in making the graph a counterexample.

Definition 2. A counterexample has the *extremal property* if it has a bad set S such that $I \setminus I_\ell \subseteq S$ and $S \cap C \subseteq C_1$. In this case, S is called an *extremal bad set*.

Lemma 4.9. *Every minimal counterexample G has the extremal property, and every locally worst bad set of G is an extremal bad set.*

Proof. Suppose for contradiction that G has a locally worst bad set S that is not an extremal bad set. By Lemma 4.7 the sets $I \cap S$, $I \setminus S$, $C \cap S$ and $C \setminus S$ are non-empty. Let u and v be the vertices in $I \cap S$ and $C \cap S$ with the highest labels, and let u' and v' be the vertices in $I \setminus S$ and $C \setminus S$ with the lowest labels respectively. Now, $S = \text{over}(u', v')$ so by Lemma 4.3 $u <_L v'$ and $v <_L u'$

If S contains non-universal vertices in C, let x be the vertex of I with the smallest label. We show that $G - x$ also is a counterexample. Furthermore, since S contains non-universal clique vertices, by Lemma 4.1 S contains at least one vertex of I_2, so x is distinct from u,u',v and v'. Also, since $x \in I_1$ and $v \notin C_1$, it follows that x is not adjacent to v or v'. Thus the rank of u,u',v and v' with respect to any set X in G is the same as the rank with respect to $X \setminus \{x\}$ in G' for each of these vertices. Let L' be the layout output by the algorithm when executed on $G - x$. By Lemma 4.3 $S \setminus \{x\}$ is covered by L'. Thus, since $\delta^{G-x}(S \setminus \{x\}) = \delta(S)$, $S \setminus \{x\}$ is a bad set for $G - x$ contradicting that G is a minimal counterexample. Thus $S \cap C \subseteq C_1$ follows.

If not $I \setminus I_\ell \subseteq S$, then let x be the vertex in I_ℓ with lowest label, and let y be the vertex in C with highest label. We show that $G' = G - xy$ also is a counterexample. First, observe that by Lemma 2.1 G' is a threshold graph. Furthermore, by the assumption that $I \setminus (I_\ell \cup S) \neq \emptyset$, x is distinct from u and u'. Also, since $S \cap C \subseteq C_1$ and G has at least three levels, y is distinct from v and v'. Let L' be the layout output by the algorithm when executed on G'. Since $\text{over}(u, v')$ and $\text{over}(u', v)$ are the same sets in G and G', and the deleted edge xy is not incident to any of u,u', v or v', $u <_{L'} v'$ and $u' <_{L'} v$ and so S is covered by L'. But $\delta^{G'}(S) = \delta(S)$ contradicting that G is a minimal counterexample. □

Lemma 4.10. *If there is a counterexample, then there is a counterexample with the extremal property and at most* 3 *levels.*

Proof. We start by showing that if there is a counterexample G on at least 4 levels with an extremal bad set S such that the sets $I \cap S$, $I \setminus S$, $C \cap S$ and $C \setminus S$ are non-empty, then there is a counterexample G' with $G \subset G'$, such that (C, I) is a threshold partition of G' and S is an extremal bad set of G'.

Let u and v be the vertices in $I \cap S$ and $C \cap S$ with the highest labels, and let u' and v' be the vertices in $I \setminus S$ and $C \setminus S$ with the lowest labels respectively. Now, $S = \text{over}(u', v')$ so by Lemma 4.3 $u <_L v'$ and $v <_L u'$. We choose x to be the vertex of I_2 with highest label and y to be the vertex if C_3 with the lowest label. We add the edge xy to G to obtain a new threshold graph G'. (C, I) is a threshold partition of G' and $G \subset G'$. Furthermore $\delta^{G'}(S') = \delta(S) \cup \{xy\}$, implying that

$$d^{G'}(S) = d(S) + 1 > \text{cw}(G) + 1 \geq \text{cw}(G \cup \{xy\}).$$

Also, in G' u' is adjacent to all vertices of C and v is a universal vertex. Let L' be the layout produced by Algorithm MinCut when run on G'. To prove that S is an extremal bad set of G' it is sufficient to show that L' covers S. However, $S = \text{over}(u', v')$ both in G and G' and none of u, u', v and v' are incident to the new edge xy so $u <_{L'} v'$ and $v <_{L'} u'$. By Lemma 4.3 L' covers S.

We can now proceed to prove the lemma. Without loss of generality, G is a minimal counterexample. By Lemma 4.9 G has an extremal locally worst bad set S. By Lemma 4.7 the sets $I \cap S$, $I \setminus S$, $C \cap S$ and $C \setminus S$ are non-empty. Thus, if G has at most 3 levels we are done, otherwise by the discussion in the previous paragraph, there is a counterexample G' with $G \subset G'$, such that (C, I) is a threshold partition of G' and S is an extremal bad set of G'. If G' has at most 3 levels we are done, otherwise we can again apply the discussion above to G' and S to get yet another counterexample G'' with $G \subset G' \subset G''$, such that (C, I) is a threshold partition of G'' and S is an extremal bad set of G''. Reiterating this argument we can continue producing counterexamples on more and more edges. Since the clique is not a counterexample, this process must stop at some point. The graph at hand at this point is a counterexample with at most 3 levels and with S as an extremal bad set. □

Definition 3. A counterexample has the *super extremal property* if it has an extremal bad set S, such that either $I_\ell \cap S \neq \emptyset$ or $S \cap C \subset C_1$. Then S is called a *super extremal bad set*.

Lemma 4.11. *There are no counterexamples with the super extremal property.*

Proof. We show that if there is a counterexample G with the super extremal property, then there is a counterexample G' with at most 2 levels. This would contradict Lemmas 4.6 and 4.8. The proof that if there is a counterexample G with the super extremal property, then there is a counterexample G' with at most 2 levels is similar to the proof of Lemma 4.10.

We start by showing that if there is a counterexample G on at least 3 levels with a super extremal bad set S such that the sets $I \cap S$, $I \setminus S$ and $C \setminus S$ are non-empty, then there is a counterexample G' with $G \subset G'$, such that (C, I) is a threshold partition of G' and S is a super extremal bad set of G'.

If $C \cap S$ is nonempty, let u and v be the vertices in $I \cap S$ and $C \cap S$ with the highest labels, and let u' and v' be the vertices in $I \setminus S$ and $C \setminus S$ with the lowest labels respectively. Now, $S = \text{over}(u', v')$ so by Lemma 4.3 $u <_L v'$ and $v <_L u'$. If $S \cap C \subset C_1$ we choose x to be the vertex of I_1 with highest label and y to be the vertex if C_2 with the lowest label. If $S \cap C \subset C_1$ does not hold, then $I_3 \cap S \neq \emptyset$ and we choose x to be the vertex of I_2 with highest label and y to be the vertex if C_3 with the lowest label. We add the edge xy to G to obtain a new threshold graph G'.

(C, I) is a threshold partition of G' and $G \subset G'$. Furthermore $\delta^{G'}(S') = \delta(S) \cup \{xy\}$, and in G' u' is adjacent to all vertices of C and v is a universal vertex. Furthermore, if $S \cap C \subset C_1$, then v' is a universal vertex both in G and G' while if $I_3 \cap S \neq \emptyset$, then u is adjacent to all vertices of C both in G and in G'. Let L' be the layout produced by Algorithm MinCut when run on G'. To prove that S is a super extremal bad set of G' it is sufficient to show that L' covers S. However, $S = \text{over}(u', v')$ both in G and G' and none of u, u', v and v' are incident to the new edge xy so $u <_{L'} v'$ and $v <_{L'} u'$. By Lemma 4.3 L' covers S.

If $C \cap S = \emptyset$, let u be the vertex in $I \cap S$ with the highest label, and let v' be the vertex in C with the lowest label. Let u' be the vertex of $I \setminus S$ with the smallest label. Observe that u' exists, as we assumed that $I \setminus S$ is nonempty. Now, $u <_L v'$ and $S = \text{over}(u', v')$. We choose x to be the vertex of I_1 with highest label and y to be the vertex of C_2 with the lowest label. We add the edge xy to G to obtain a new threshold graph G'. (C, I) is a threshold partition of G' and $G \subset G'$. Furthermore $\delta^{G'}(S') = \delta(S) \cup \{xy\}$, and in G', u is adjacent to all vertices of C and v' is a universal vertex. Let L' be the layout L' produced by Algorithm MinCut when run on G'. To prove that S is a super extremal bad set of G' it is sufficient to show that L' covers S. However, $S = \text{over}(u', v')$ both in G and in G', and none of u, u' and v' are incident to the new edge xy so $u <_{L'} v'$. Thus L' covers S.

We can now proceed to show that if there is a counterexample G with the super extremal property, then there is a counterexample G' with at most 2 levels. If G has at most 2 levels we are done, so assume that G has at least 3 levels. Let $S = V_i$ be the largest super extremal bad set of G. By the definition of the extremal property $I \cap S$ and $C \setminus S$ are nonempty. Also, if a is the vertex of I with the largest label and b is the vertex of C_2 with highest label, then by Lemma 4.2 $b <_L a$. Since $b \notin S$ we have $a \notin S$ which implies $I \setminus S \neq \emptyset$.

By the discussion in the first paragraphs of the proof, there is a counterexample G' with $G \subset G'$, such that (C, I) is a threshold partition of G' and S is a super extremal bad set of G'. If G' has at most 2 levels we are done, otherwise we can again apply the discussion above to G' and S to get yet another counterexample G'' with $G \subset G' \subset G''$, such that (C, I) is a threshold partition of G'' and S is a super extremal bad set of G''. Reiterating this argument we can continue producing counterexamples on more and more edges. Since a clique is not a counterexample, this process must stop at some point. The graph G^* we are considering at this step is a counterexample with at most 2 levels, contradicting Lemmas 4.6 and 4.8. □

Lemmas 4.9, 4.10, and 4.11 allow us to concentrate on counterexamples on exactly 3 levels with the extremal property, but without the super extremal property.

Definition 4. We say that a counterexample with 3 levels and the extremal property has the *snake property* if (i) $u <_L v$, where u is the vertex of I_2 with highest label and v is the vertex of C_1 with highest label, and (ii) $u' <_L v'$, where u' is the vertex of I_3 with lowest label and v' is the vertex of C_2 with lowest label.

Lemma 4.12. *If there is a counterexample with* 3 *levels with the extremal property, then there is a counterexample with* 3 *levels with the extremal and the snake properties.*

Proof. Let G be a counterexample with the extremal property, with 3 levels. Let S be an extremal bad set of G. By Lemma 4.11 S is not super extremal. Thus $S = I_1 \cup I_2 \cup C_1$. Let u be the vertex of I_2 with highest label and v be the vertex of C_1 with highest label. Now, let u' and v' be the vertices of I_3 and C_2 with lowest labels respectively. If $u <_L v$ and $u' <_L v'$

we are done, so either $v <_L u$ or $v' <_L u'$ must hold. We choose x to be the vertex of I_1 with lowest label, and let $G' = G - x$. Let L' be the layout constructed by the algorithm when run on G'. We show that L' covers $S' = S \setminus \{x\}$. Since x is nonadjacent to both u and v', Lemma 4.2 implies $u <_{L'} v'$.

Suppose that $v' <_L u'$. Since x is nonadjacent to both u' and v', Lemma 4.2 implies $v' <_{L'} u'$. Since $v <_{L'} v'$ it follows that $v <_{L'} u'$. By Lemma 4.3, L' covers S'. Suppose now that $v <_L u$. Then in G, $\text{rank}_{\text{over}(u,v)}(v) \leq \text{rank}_{\text{over}(u,v)}(u)$. Since u is adjacent to v but not to u' and all neighbors of u' that are non-neighbors of u are in C_3, we have that $\text{rank}_{\text{over}(u',v)}(v) + 2 \leq \text{rank}_{\text{over}(u,v)}(v) \leq \text{rank}_{\text{over}(u,v)}(u) \leq \text{rank}_{\text{over}(u',v)}(u')$. Deleting x increases the rank of v by one and does not decrease the rank of u', so by Lemma 4.2, $v <_{L'} u'$. By Lemma 4.3, L' covers S'

Now, $\delta^{G'}(S') = \delta(S)$ so S' is a bad set of G'. By Lemma 4.8 G' has 3 levels. Also S' is an extremal bad set of G'. If G' does not have the snake property, we can apply the argument above to get a counterexample G'' on 3 levels with the extremal property and even fewer vertices than G'. Reiterating this argument we can continue producing counterexamples on fewer and fewer vertices. Since the single vertex graph is not a counterexample, this process must stop at some point. The graph G^* we are considering at this step is a counterexample with 3 levels and the extremal and snake properties. □

Lemma 4.13. *In a counterexample with 3 levels with the extremal and snake properties, n is even, $|C|$ and $|I|$ are odd, $|C_1| = |C_2| + |C_3| + 1$, $|I_3| = |I_1| + |I_2| + 1$, and $|I_1| + |I_2| + |C_1| = \frac{n}{2}$.*

Proof. Let G be a counterexample with 3 levels with the extremal and snake properties. Let S be an extremal bad set of G. By Lemma 4.11, S is not super extremal, so $S = V_i = I_1 \cup I_2 \cup C_1$. Let u and v be the vertices of I_2 and C_1 with highest labels, and let u' and v' be the vertices of I_3 and C_2 with lowest labels respectively. Since S has the snake property, $v = v_i$ and $u' = v_{i+1}$. If $\text{rank}_{V_{i-1}}(v) \leq 0$, then V_{i-1} is a super extremal bad set, contradicting Lemma 4.11. Similarly, if $\text{rank}_{V_i}(u') \geq 0$, then V_{i+1} is a super extremal bad set, contradicting Lemma 4.11. Thus $\text{rank}_{V_{i-1}}(v) > 0$ and $\text{rank}_{V_i}(u') < 0$. By Observation 4.4 $\text{rank}_{V_{i-1}}(v) = 1$ and $\text{rank}_{V_i}(u') = -1$. Since v is a universal vertex this implies that n is even and that $|S| = |I_1| + |I_2| + |C_1| = \frac{n}{2}$. Since u' is adjacent to all vertices of the clique, $\text{rank}_{V_i}(u') = |C_2| + |C_3| - |C_1| = -1$ so $|C_1| = |C_2| + |C_3| + 1$ and $|C|$ is odd. Since $|I| + |C| = n$ is even, $|I|$ is odd. Finally, since $|I_1| + |I_2| + |C_1| = |I_3| + |C_2| + |C_3|$ and $|C_1| = |C_2| + |C_3| + 1$ we have $|I_3| = |I_1| + |I_2| + 1$. □

At this point all that remains is to analyze how a counterexample on 3 levels and the extremal and snake properties looks, and to show that in such a graph G, $\text{cw}_L(G) \leq \text{bw}(G) \leq \text{cw}(G)$. Now we are ready to show our main result.

Theorem 4.14. *For any threshold graph G, $cw_L(G) = cw(G)$.*

Proof. Suppose for contradiction that there is a counterexample. Then, by Lemmas 4.6, 4.8, 4.9, 4.10 and 4.12 there is a counterexample G on 3 levels with the snake and extremal properties. Since Lemma 4.11 implies that G does not have the super extremal property, $I_1 \cup I_2 \cup C_1$ is a bad set of G. By Lemma 4.13, n is even, $|C|$ and $|I|$ are odd, $|C_1| = |C_2| + |C_3| + 1$, $|I_3| = |I_1| + |I_2| + 1$, and $|I_1| + |I_2| + |C_1| = \frac{n}{2}$.

Let S be a vertex set on $\frac{n}{2}$ vertices that minimizes $d(S)$, that is, with $d(S) = \text{bw}(G)$. Notice that $\delta(S) = \delta(V \setminus S)$. Thus, without loss of generality $|S \cap I| > \frac{|I|}{2}$. We view the set S as a set

of $\frac{n}{2}$ pebbles that have been placed on distinct vertices of G. We can move pebbles from I_3 to C_1 and keep optimality of S, unless one of the following is true:

(i) there are no pebbles on vertices of I_3,

(ii) all vertices of C_1 have pebbles on them,

(iii) moving a pebble from a vertex in I_3 to a vertex in C_1 increases $d(S)$.

Moving a pebble from a vertex x to a vertex y increases $d(S)$ by $\text{rank}_{S\setminus\{x\}}(y) - \text{rank}_{S\setminus\{x\}}(x)$. Thus, if there is a pebble on a vertex x in I_3 and a free spot on a vertex y in C_1, moving a pebble from x to y does not increase $d(S)$ if and only if

$$\text{rank}_{S\setminus\{x\}}(y) - \text{rank}_{S\setminus\{x\}}(x) = 1 - (|C \setminus S| - |C \cap S|) \leq 0.$$

Rearranging terms yields that moving a pebble from x to y does not increase $d(S)$ if and only if $|C \cap S| < |C \setminus S|$. In addition, one should notice that if there are no pebbles on vertices of I_3, then $|C \cap S| > |C \setminus S|$ because $|I_3| = |I_1| + |I_2| + 1$. Similarly, if all vertices of C_1 have pebbles on them, then $|C \cap S| > |C \setminus S|$ because $|C_1| = |C_2| + |C_3| + 1$.

By our choice of S, before we start moving any pebbles, $|I \cap S| > |I \setminus S|$. Since $|S| = \frac{n}{2}$ this means that $|C \cap S| < |C \setminus S|$. Therefore, by the discussion in the previous paragraph we can move pebbles from I_3 to C_1, preserving minimality of $d(S)$ until the inequality flips from $|C \cap S| < |C \setminus S|$ to $|C \cap S| > |C \setminus S|$. At this point,

$$|C \cap S| = \left\lceil \frac{|C|}{2} \right\rceil = |C_1| \quad \text{and} \quad |I \cap S| = |I_1| + |I_2|.$$

Let α, β and γ be the ranks of a vertex in I_1, I_2 and I_3 with respect to S.

If $\alpha \leq \gamma$ and $\beta \leq \gamma$ we can move pebbles from I_3 to I_1 and I_2 keeping optimality of S. Since exactly $|I_1| + |I_2|$ pebbles are placed on vertices of I, after the move every vertex of $I_1 \cup I_2$ has a pebble on it, and no pebbles are on vertices in I_3. Now we can safely move all pebbles in C_2 and C_3 to C_1, keeping optimality of S. Since exactly $|C_1|$ pebbles are placed on vertices of C, after the move every vertex of C_1 has a pebble on it, and no pebbles are on vertices in $C_2 \cup C_3$. But this means that $S = I_1 \cup I_2 \cup C_1$ and $d(S) \leq \text{bw}(G) \leq \text{cw}(G)$ contradicting that $I_1 \cup I_2 \cup C_1$ is a bad set of G.

If $\alpha \geq \gamma$ and $\beta \geq \gamma$ we can move all pebbles from I_1 and I_2 to I_3 keeping optimality of S. Since exactly $|I_1| + |I_2|$ pebbles are placed on vertices of I, after this move all but one vertex of I_3 has a pebble on it, and no pebbles are on vertices in $I_1 \cup I_2$. Now we can safely move pebbles in C_1 to C_2 and C_3, keeping optimality of S. Since exactly $|C_1|$ pebbles are placed on vertices of C, after this move exactly one vertex of C_1 has a pebble on it, and all vertices of $C_2 \cup C_3$ have pebbles on them. Let x be the vertex in I_3 without a pebble and y be the vertex in C_1 with a pebble. After the moves, $\text{rank}_{S\setminus\{y\}}(x) = \text{rank}_{S\setminus\{y\}}(y) = 1$, so we can move the pebble on y to x, keeping optimality of S. However $d(I_1 \cup I_2 \cup C_1) = d(V \setminus S) = d(S) \leq \text{bw}(G)$ contradicting that $I_1 \cup I_2 \cup C_1$ is a bad set of G.

If $\alpha \leq \gamma \leq \beta$ we can move pebbles from I_2 and I_3 to I_1 maintaining optimality until each vertex of I_1 has a pebble on it. If any pebbles remain in I_2 we can move these pebbles to I_3. Since there are $|I_1| + |I_2|$ pebbles in I there are exactly $|I_2|$ vertices in I_3 that have pebbles on them. Now, we can move all pebbles in C_2 to C_3 and C_1 maintaining optimality until there are no pebbles left in C_2. If $|I_1| \geq |I_2|$ we can move all pebbles from C_3 to C_1 maintaining

optimality. After this move, the set of vertices in C with pebbles on them is exactly C_1. Thus we can move all pebbles in I_3 to I_2 maintaining optimality. In this case, $S = I_1 \cup I_2 \cup C_1$, but $d(S) \leq \text{bw}(G)$ contradicting that $I_1 \cup I_2 \cup C_1$ is a bad set of G.

If $|I_1| < |I_2|$ we can move pebbles from C_1 to C_3 until all vertices of C_3 have pebbles on them. After this move there are exactly $|C_1| - |C_3|$ pebbles on vertices in C_1. We consider $d(S)$ and compare it to

$$d(I_1 \cup I_2 \cup C_1) = |C_1|(|C_2| + |C_3|) + |I_2||C_2| + |I_3||C_1|.$$

Counting the edges of $d(S)$ we obtain

$$d(S) = |C_1|(|C_2| + |C_3|) + |I_1||C_3| + |I_2|(|C_1| - |C_3|) + (|I_3| - |I_2|)|C_1| + |I_2|(|C_2| + |C_3|).$$

Simplifying yields

$$d(S) = |C_1|(|C_2| + |C_3|) + |I_1||C_3| + |I_3||C_1| + |I_2||C_2|.$$

But this means that $d(I_1 \cup I_2 \cup C_1) < d(S) \leq \text{bw}(G)$ contradicting that S is a bad set of G.

Finally, suppose $\alpha \geq \gamma \geq \beta$. Since $d(S) = d(V \setminus S)$ we can move all pebbles over to vertices that do not have pebbles and preserve optimality. There are now exactly $|I_3|$ pebbles in I and $|C_2| + |C_3|$ pebbles in $|C|$. Since $|I_3| > |I_1| + |I_2|$ there is a pebble on a vertex x in I_3. Also, since $|C_1| > |C_2| + |C_3|$ there is a vertex y with no pebble in C_1. At this point, $\text{rank}_{S\setminus\{x\}}(x) = \text{rank}_{S\setminus\{x\}}(y) = 1$, so we can move a pebble from x to y and again obtain a set S with pebbles on $|I_1|+|I_2|$ vertices in I and $|C_1|$ vertices in C. In addition if α', β' and γ' are the ranks of a vertex in I_1, I_2 and I_3 with respect to S, then

$$\alpha' = -\alpha - 2,\ \beta' = -\beta - 2 \ \text{ and } \gamma' = -\gamma - 2.$$

Thus $\alpha' \leq \gamma' \leq \beta'$ and the discussion in the previous paragraphs applies. This concludes the proof. □

Theorem 4.15. *The cutwidth of a threshold graph G on n vertices can be computed in $O(n)$ time if the threshold partition of G is given as input, and $O(n + m)$ time if the adjacency list representation of G is given.*

Proof. Given the adjacency list representation of G, a threshold partition can be computed in $O(n+m)$ time [12]. We now describe an implementation of Algorithm MinCut that runs in $O(n)$ time if the threshold partition of G is given. Let $(C_1 \cup C_2 \cdots \cup C_\ell, I_1 \cup I_2 \cdots \cup I_\ell)$ be the given threshold partition of G. By Lemma 4.1 we know that Algorithm MinCut picks vertices of I by increasing label and the vertices of C by increasing label. Therefore we keep track of the not yet picked vertices $u \in I$ and $v \in C$ with the lowest labels. We also keep track of the ranks of u and v with respect to $\text{over}(u, v)$, that is $r_u = \text{rank}_{\text{over}(u,v)}(u)$ and $r_v = \text{rank}_{\text{over}(u,v)}(u)$. At each step of the algorithm we pick the one of u and v with the lowest rank (and highest degree if their rank is equal, lowest label if both rank and degree is equal). We now need to update the variables u, v, r_u and r_v. Lemma 4.1 guarantees that u and v are adjacent, so if we pick u we reduce r_v by 2 and if we pick v we reduce both r_u and r_v by 2. Finally, if we picked u, we need to correct r_u for the fact that the next vertex in I could be in a higher level, and similarly we need to correct r_v. If the algorithm picked u, let u' be the vertex in I with $\text{label}(u)+1 = \text{label}(u')$. If $\text{level}(u') > \text{level}(u)$ we increase r_u by $|C_{\text{level}(u')}|$ because Lemma 4.1 guarantees that no vertices

of $C_{\text{level}(u')}$ have been picked yet. Similarly, if the algorithm picked v, let v' be the vertex in C with $\text{label}(v) + 1 = \text{label}(v')$. If $\text{level}(v') > \text{level}(v)$ we increase r_v by $|I_{\text{level}(v)}|$ because Lemma 4.1 guarantees that all vertices of $I_{\text{level}(v)}$ have already been picked. For each new vertex to be picked the algorithm does $O(1)$ work so the total time complexity is $O(n)$. □

5 Concluding remarks: cutwidth of interval graphs

A natural open question and a future research direction is resolving the computational complexity of cutwidth on interval graphs. A graph is *interval* if sets of consecutive integers (intervals) can be assigned to its vertices such that two vertices are adjacent if and only if their intervals overlap. Some inherently difficult graph problems, like bandwidth, are polynomially solvable on interval graphs [16], whereas others, like optimal linear arrangement, are NP-complete [7]. Optimal linear arrangement can be seen as sum-bandwidth or sum-cutwidth, equivalently (see [10] for definitions).

A subclass of interval graphs and a superclass of threshold graphs is the class of trivially perfect graphs. Extending our results even to trivially perfect graphs seems to be a non-trivial open problem.

Simple examples exist to show that Algorithm MinCut can produce a layout with cutwidth that is a factor of $O(n)$ larger than $\text{cw}(G)$ when G is an interval graph, or even a proper interval graph. An interval graph is *proper interval* if it has an interval model where no interval properly contains another. Interestingly, for proper interval graphs an ordering of the vertices of the input graph by increasing right endpoint of their corresponding intervals is a minimum cutwidth layout [25]. Note that an increasing right endpoint order is not necessarily an optimal layout for a threshold graph; a star is a simple counterexample.

Acknowledgements

The authors would like to express their thanks to the anonymous referees whose suggestions helped improve the presentation of the paper.

References

[1] D. Adolphson and T. C. Hu. Optimal linear ordering. *SIAM J. Appl. Math.* 25: 403–423, 1973.

[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 FOCS 1996*, pp. 21–30, IEEE.

[3] G. Blin, G. Fertin, D. Hermelin, and S. Vialette. Fixed-parameter algorithms for protein similarity search under mRNA structure constraints. *Journal of Discrete Algorithms*, 6:618–626, 2008.

[4] R. A. Botafogo. Cluster analysis for hypertext systems. In *Proceedings of SIGIR 1993*, pp. 116–125, ACM.

[5] A. Brandstädt, V. B. Le, and J. Spinrad. *Graph Classes: A Survey.* SIAM, Philadelphia, 1999.

[6] M.J. Chung, F. Makedon, I.H. Sudborough, and J. Turner. Polynomial time algorithms for the min cut problem on degree restricted trees. *SIAM Journal on Computing*, 14:158–177, 1985.

[7] J. Cohen, F. V. Fomin, P. Heggernes, D. Kratsch, and G. Kucherov. Optimal linear arrangement of interval graphs. In *Proceedings of MFCS 2006*, LNCS 4162, pages 267–279, Springer.

[8] F. Gavril. Some NP-complete problems on graphs. In *11th Conference on Information Sciences and Systems*, John Hopkins University, Baltimore, 91–95, 1977.

[9] J. Diaz, M. Penrose, J. Petit, and M. Serna. Approximating layout problems on random geometric graphs. *Journal of Algorithms*, 39:78–117, 2001.

[10] J. Diaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34:313–356, 2002.

[11] S. Földes and P. L. Hammer. Split graphs. *Congressus Numerantium*, 19:311–315, 1977.

[12] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Second edition. Annals of Discrete Mathematics 57. Elsevier, 2004.

[13] P. Heggernes and C. Papadopoulos. Single-edge monotonic sequences of graphs and linear-time algorithms for minimal completions and deletions. *Theoretical Computer Science*, 410:1–15, 2009.

[14] M. Junguer, 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.

[15] D. R. Karger. A randomized fully polynomial approximation scheme for the all-terminal network reliability problem. *SIAM Journal on Computing*, 29:492–514, 1999.

[16] D. J. Kleitman and R. V. Vohra. Computing the bandwidth of interval graphs. *SIAM Journal on Disc. Math.*, 3:373–375, 1990.

[17] F.T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46:787–832, 1999.

[18] N. Mahadev and U. Peled. *Threshold graphs and related topics*. Annals of Discrete Mathematics 56. North Holland, 1995.

[19] F. Makedon and I. H. Sudborough. On minimizing width in linear layouts. *Discrete Applied Mathematics*, 23:243–265, 1989.

[20] B. Monien and I. H. Sudborough. Min cut is NP-complete for edge weighted trees. *Theoretical Computer Science* 58:209–229, 1988.

[21] P. Mutzel. A polyhedral approach to planar augmentation and related problems. In *Proceedings of ESA 1995*, LNCS 979, pp. 497–507, Springer.

[22] D. M. Thilikos, M. J. Serna, and H. L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *Journal of Algorithms*, 56:1–24, 2005.

[23] D. M. Thilikos, M. J. Serna, and H. L. Bodlaender. Cutwidth II: Algorithms for partial w-trees of bounded degree. *Journal of Algorithms*, 56:24–49, 2005.

[24] M. Yannakakis. A polynomial algorithm for the min cut linear arrangement of trees. *Journal of the ACM*, 32:950–988, 1985.

[25] J. Yuan and S. Zhou. Optimal labelling of unit interval graphs. *Appl. Math. J. Chinese Univ. Ser. B* (English edition), 10:337–344, 1995.

Appendix

Here we give two examples of the Algorithm MinCut when applied on graphs relative to threshold graphs. More precisely we present a chain graph in Figure 3 and a trivially-perfect graph in Figure 4; proper definitions of such families related to threshold graphs can be found in [12]. In both cases we give two layouts: one layout that the algorithm produces and another one of strictly smaller cutwidth. Therefore our algorithm cannot be directly applied on larger or related classes of threshold graphs.

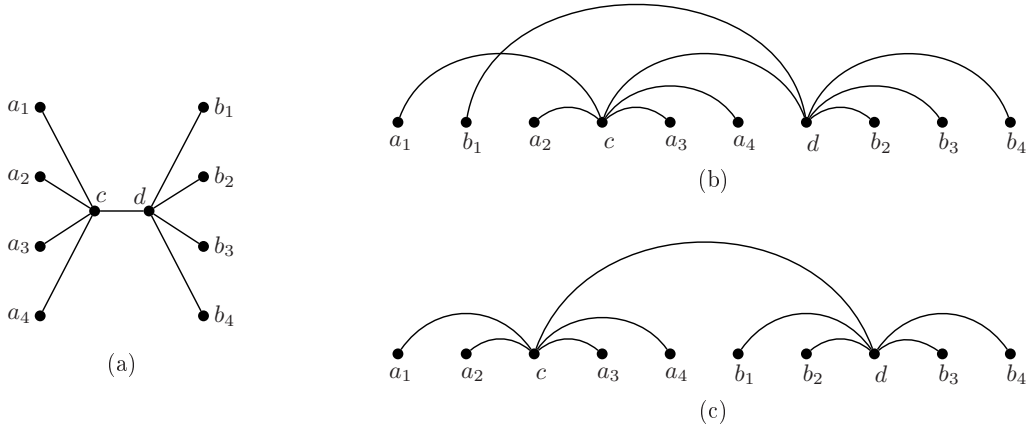


Figure 3: (a) A chain graph, (b) a layout of cutwidth 4 computed by the Algorithm MinCut, and (c) a layout of cutwidth 3.

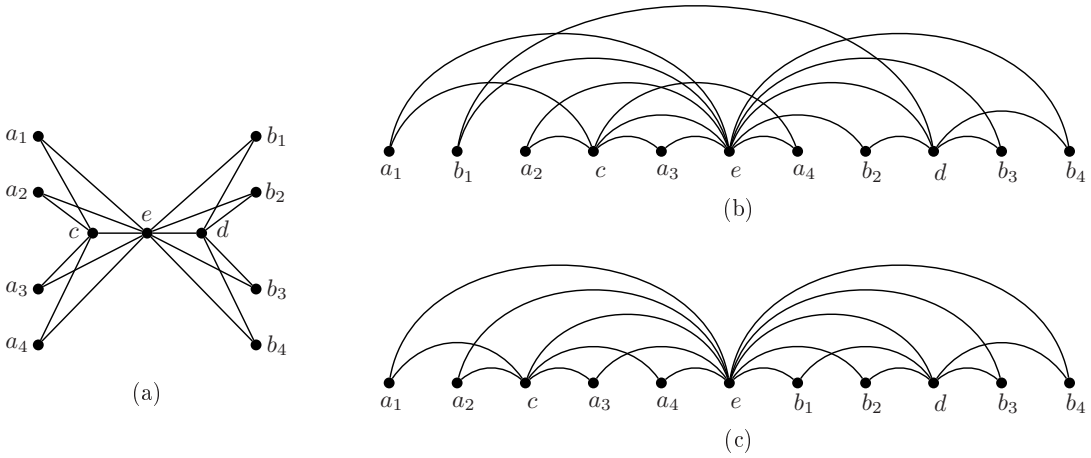


Figure 4: (a) A trivially-perfect graph, (b) a layout of cutwidth 7 computed by the Algorithm MinCut, and (c) a layout of cutwidth 5.