REPORTS IN INFORMATICS

ISSN 0333-3590

Single-edge monotonic sequences of graphs and linear-time algorithms for minimal completions and deletions

Pinar Heggernes Charis Papadopoulos

REPORT NO 345 February 2007

Department of Informatics

UNIVERSITY OF BERGEN

Bergen, Norway

This report has URL http://www.ii.uib.no/publikasjoner/texrap/pdf/2007-345.pdf

Reports in Informatics from Department of Informatics, University of Bergen, Norway, is available at http://www.ii.uib.no/publikasjoner/texrap/.

Requests for paper copies of this report can be sent to:

Department of Informatics, University of Bergen, Høyteknologisenteret,
P.O. Box 7800, N-5020 Bergen, Norway

Single-edge monotonic sequences of graphs and linear-time algorithms for minimal completions and deletions*

Pinar Heggernes† Charis Papadopoulos†

Abstract

We study graph properties that admit an increasing, or equivalently decreasing, sequence of graphs on the same vertex set such that for any two consecutive graphs in the sequence their difference is a single edge. This is useful for characterizing and computing minimal completions and deletions of arbitrary graphs into having these properties. We prove that threshold graphs and chain graphs admit such sequences. Based on this characterization and other structural properties, we present linear-time algorithms both for computing minimal completions and deletions into threshold, chain, and bipartite graphs, and for extracting a minimal completion or deletion from a given completion or deletion. Minimum completions and deletions into these classes are NP-hard to compute.

1 Introduction

A graph property $\mathcal{P}$ is called *monotone* if it is closed under any edge or vertex removal. Equivalently, a property is monotone if it is closed under taking subgraphs that are not necessarily induced. A property is *hereditary* if it is closed under taking induced subgraphs. Every monotone property is hereditary but the converse is not true. For example bipartiteness and planarity are monotone properties, as they are characterized through forbidden subgraphs that are not necessarily induced, whereas perfectness is a hereditary but not monotone property. Some of the most well-studied graph properties are monotone [1, 3] or hereditary [17]. If a given monotone (hereditary) property is equivalent to belonging to a graph class then this graph class is called monotone (hereditary).

In this paper, we introduce *sandwich monotonicity* of graph properties and graph classes. We say that a graph property $\mathcal{P}$ is *sandwich monotone* if $\mathcal{P}$ satisfies the following: Given two graphs $G_1 = (V, E)$ and $G_2 = (V, E \cup F)$ that both satisfy $\mathcal{P}$, the edges in F can be ordered $f_1, f_2, ..., f_{|F|}$ such that when single edges of F are added to G_1 one by one in this order (or removed from G_2 in the reverse order), the graph obtained at each step satisfies $\mathcal{P}$. Every monotone property is clearly sandwich monotone as well. However every hereditary property is not necessarily sandwich monotone. Here, we are interested in identifying non-monotone hereditary graph classes that are sandwich monotone. Until now, sandwich monotonicity of only two such classes have been shown: chordal graphs [34] and split graphs [18], and it has been an open question which other graph such classes are sandwich monotone [2]. Simple examples exist to show that the hereditary classes of perfect graphs, comparability graphs, permutation graphs, cographs, trivially perfect graphs, and interval graphs are not sandwich monotone.[1] In this paper we show that threshold graphs and chain graphs are sandwich monotone.

A motivation for studying sandwich monotonicity is from dynamic graph algorithms [24, 35]. Many dynamic algorithms for maintaining a property do not allow the addition or deletion of edges unless the property is still satisfied after this modification. Assume that we are given an initial graph and

*This work is supported by the Research Council of Norway through grant 166429/V30.

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

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

a set of edges which we want to add to this graph in a dynamic fashion one by one (and do some other computations at each step). If the given graph and the graph where all the given edges are added satisfy a sandwich monotone property, then there is an order of the edges so that we can use a dynamic algorithm for maintaining this property to add these edges one by one. If the property that the dynamic algorithm maintains is not sandwich monotone, then we will reach a point where we cannot add any single edge of the given set although we know that adding all edges acquires back the property (see [35]).

Our main motivation for studying sandwich monotonicity comes from the problem of adding edges to or deleting edges from a given arbitrary graph so that it satisfies a desired property. For example, a *chordal completion* is a chordal supergraph on the same vertex set, and a *bipartite deletion* is a spanning bipartite subgraph of an arbitrary graph. Completions and deletions into other graph classes are defined analogously. A completion (deletion) is *minimum* if it has the smallest possible number of added (deleted) edges. The problems of computing minimum completions or deletions are applicable in several areas such as molecular biology, numerical algebra and, more generally, to areas involving graph modeling with some missing edges due to lacking data [16, 29, 33]. Unfortunately minimum completions and deletions into most interesting graph classes, including threshold, chain, and bipartite graphs, are NP-hard to compute [15, 7, 26, 29, 37]. However, minimum completions (deletions) are a subset of minimal completions (deletions), and hence we can search for minimum among the set of minimal. A completion (deletion) is *minimal* if no subset of the added (deleted) edges is enough to give the desired property when added to (deleted from) the input graph.

Given as input an arbitrary graph, there are are two problems related to minimal completions (deletions) : 1. Computing a minimal completion (deletion) of the given input graph into the desired graph class, 2. Extracting a minimal completion (deletion) which is a subgraph (supergraph) of a given arbitrary completion (deletion) of the input graph into the desired class. A solution for problem 2 requires a characterization of minimal completions (deletions) into a given class, and readily gives a solution of problem 1. A solution for problem 1 might generate only a subset of all possible minimal completions (deletions), and does not necessarily solve problem 2. Solution of problem 2 in polynomial time is known only for completions into chordal [4, 10], split [18], and interval [21] graphs, and for deletions into chordal [11], split [19], and planar [12, 23] graphs. Characterizations of minimal chordal completions [34, 30, 6] have in addition made it possible to design approximation algorithms [29] and fast exact exponential time algorithms [13] for computing minimum chordal completions. A solution of problem 2 also allows the computation of minimal completions that are not far from minimum, since one can generate first a completion by an established heuristic or approximation algorithm for the minimization problem, and then extract a minimal completion that has fewer edges [5].

For a graph property $\mathcal{P}$ that is sandwich monotone, problem 2 of extracting a minimal completion from a given completion of an input graph into $\mathcal{P}$ has always a polynomial time solution if $\mathcal{P}$ can be recognized in polynomial time. Given G and a supergraph G_2 of G satisfying $\mathcal{P}$, if G_2 is not a minimal completion, then a minimal completion G_1 exists sandwiched between G and G_2. Hence by trying one by one all edges of G_2 that are not in G for removal, one obtains a minimal extraction algorithm with a number of iterations that is quadratic in the number of edges appearing in G_2 but not in G.

In this paper, by showing that threshold graphs and chain graphs are sandwich monotone, we thus establish that minimal completions of arbitrary graphs into these graph classes can be computed in polynomial time. Even more interesting, we are able to give linear-time algorithms for minimal completions and deletions into these graph classes, minimal deletions into bipartite graphs, and minimal completions into co-bipartite graphs. This does not follow from sandwich monotonicity in general. Furthermore, we solve the problem of extracting a minimal completion or deletion from a given one (problem 2 above) in linear time for all these graph classes. Linear-time minimal completion algorithms have been known only into two classes previously; split [18] and proper interval [32] graphs. The only linear-time minimal extraction algorithm known is the one into split graphs [18]. Linear-time minimal deletion algorithms are known only into split [19] and planar graphs (which are monotone) [12, 23].

2 Preliminaries

We consider undirected finite graphs with no loops or multiple edges. For a graph $G = (V, E)$, we denote its vertex and edge set by $V(G) = V$ and $E(G) = E$, respectively, with $n = |V|$ and $m = |E|$. 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 $(V, E \setminus \{uv\})$.

The *neighborhood* of a vertex x of G is $N_G(x) = \{v \mid xv \in E\}$. The *closed neighborhood* of x is defined as $N_G[x] = N_G(x) \cup \{x\}$. The *degree* of a vertex x in a graph G, denoted by $d_G(x)$, is the number of edges incident to x; thus, $d_G(x) = |N_G(x)|$. If $S \subseteq V$, then the neighbors of S, denoted by $N_G(S)$, are given by $\bigcup_{x \in S} N_G(x) \setminus S$. We will omit the subscript 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 connected subgraph of it with a maximal set of vertices and edges. A vertex x of G is *universal* if $N_G[x] = V$ and is *isolated* if it has no neighbors in G. A *clique* is a set of pairwise adjacent vertices, while an *independent set* is a set of pairwise non-adjacent vertices. The size of a largest clique in G is denoted by $\omega(G)$.

A *chord* of a path or a cycle is an edge between two non-consecutive vertices of the path or the cycle. A chordless cycle on k vertices is denoted by C_k and a chordless path on k vertices is denoted by P_k. The graph consisting of only two disjoint edges is denoted by $2K_2$.

For a pair of vertices x, y of G we call xy a *non-edge* of G if $xy \notin E$. The *complement* $\overline{G}$ of a graph G consists of all vertices and all non-edges of G. For a graph class $\mathcal{C}$, the class *co-$\mathcal{C}$* is the set of graphs $\overline{G}$ for which $G \in \mathcal{C}$. A class $\mathcal{C}$ is *self-complementary* if $\mathcal{C} = \text{co-}\mathcal{C}$.

Given an arbitrary graph $G = (V, E)$ and a graph class $\mathcal{C}$, a $\mathcal{C}$ *completion* of G is a graph $H = (V, E \cup F)$ such that $H \in \mathcal{C}$. Similarly, $H = (V, E \setminus D)$ is a $\mathcal{C}$ *deletion* of G if $H \in \mathcal{C}$. The edges added to the original graph in order to obtain a completion are called *fill edges*, and the edges removed from the original graph in order to obtain a deletion are called *deleted edges*.

Speaking about $\mathcal{C}$ completions or $\mathcal{C}$ deletions of input graphs is only meaningful if every allowed input graph can be embedded in a graph of $\mathcal{C}$ by adding or removing edges, respectively. For example, if complete graphs belong to $\mathcal{C}$ then any graph has a $\mathcal{C}$ completion, and if edgeless graphs belong to $\mathcal{C}$ then any graph has a $\mathcal{C}$ deletion.

3 Minimal completions and deletions into sandwich monotone graph classes

We start by giving a proper definition of the new monotonicity measure on graph classes.

Definition 3.1. A graph class $\mathcal{C}$ is *sandwich monotone* if the following is true for any pair of graphs $G = (V, E)$ and $H = (V, E \cup F)$ in $\mathcal{C}$ with $E \cap F = \emptyset$: There is an ordering $f_1, f_2, \ldots, f_{|F|}$ of the edges in F such that in the sequence of graphs $G = G_0, G_1, \ldots, G_{|F|} = H$, where G_{i-1} is obtained by removing edge f_i from G_i, (or equivalently, G_i is obtained by adding edge f_i to G_{i-1}), every graph belongs to $\mathcal{C}$.

Minimal completions and deletions into sandwich monotone graph classes have the following algorithmically useful characterization, as observed on chordal graphs [34] and split graphs [18] previously.

Observation 3.2. *Let $\mathcal{C}$ be a graph class and let $\mathcal{P}$ be the property of belonging to $\mathcal{C}$. The following are equivalent:*

(i) $\mathcal{C}$ is sandwich monotone.

(ii) co-$\mathcal{C}$ is sandwich monotone.

(iii) A $\mathcal{C}$ completion is minimal if and only if no single fill edge can be removed without destroying the property $\mathcal{P}$.

(iv) A $\mathcal{C}$ deletion is minimal if and only if no single deleted edge can be added without destroying the property $\mathcal{P}$.

Proof. Statements (i) and (ii) are equivalent by Definition 3.1. For statement (iii), if a $\mathcal{C}$ completion is minimal then no subset of its fill edges can be removed so no single fill edge can be removed either. For the other direction, if a $\mathcal{C}$ completion G_2 of a given graph G is not minimal then another $\mathcal{C}$ completion G_1 of G exists such that $G \subseteq G_1 \subset G_2$. Since both G_1 and G_2 belong to $\mathcal{C}$ and $\mathcal{C}$ is sandwich monotone, there is a fill edge that can be removed from G_2 without destroying $\mathcal{P}$. Same arguments can be used for proving (iv). □

The next observation shows that extraction of a minimal completion or deletion from a given completion or deletion can be done in polynomial time for sandwich monotone graph classes that can be recognized in polynomial time. This corresponds to the solution of problem 2 mentioned in the introduction, in polynomial time. Furthermore, it implies that given only the input graph, a minimal completion or deletion can be computed in polynomial time, too, since we can always start with a trivial completion or deletion, like a complete graph or an edgeless graph.

Observation 3.3. *Let $\mathcal{C}$ be a sandwich monotone graph class. Given a polynomial time algorithm for the recognition of $\mathcal{C}$, there is a polynomial time algorithm for extracting a minimal $\mathcal{C}$ completion (deletion) of an arbitrary graph G from any given $\mathcal{C}$ completion (deletion) of G.*

Proof. Let $H = (V, E \cup F)$ be any given $\mathcal{C}$ completion of an input graph $G = (V, E)$ with $E \cap F = \emptyset$. For every edge $f \in F$ check in polynomial time whether $H - f$ belongs to $\mathcal{C}$. If yes, remove this fill edge, and repeat the process as long as there are more edges left in F. If no, conclude that a minimal $\mathcal{C}$ completion is reached by case (iii) of Observation 3.2. This will require $O(|F|^2)$ iterations, and each iteration requires polynomial time. For deletions, let $H = (V, E \setminus D)$ be any given $\mathcal{C}$ deletion of G with $D \subseteq E$. For every edge $d \in D$ check in polynomial time whether $H + d$ belongs to $\mathcal{C}$. If yes, add this deleted edge back, and repeat the process as long as there are more edges left in D. If no, conclude that a minimal $\mathcal{C}$ deletion is reached by case (iv) of Observation 3.2. This will require $O(|D|^2)$ iterations, and each iteration is polynomial. □

Note that although extraction of a minimal completion requires $O(|F|^2)$ iterations, deciding whether a given completion is minimal requires only $O(|F|)$ iterations since we can stop as soon as we have found one removable fill edge, or have scanned whole F.

Even though we know that $\mathcal{C}$ completions and deletions can be computed in polynomial time for a sandwich monotone graph class $\mathcal{C}$, the running time described in the proof of Observation 3.3 is not practical. In the following sections, we will give linear-time algorithms for computing and extracting minimal completions and deletions into threshold, bipartite, and chain graphs.

For the sandwich monotone graph classes previously studied for completions and deletions, linear-time algorithms exist for computing and extracting minimal split completions [18] and deletions [19], and minimal planar deletions [12, 23]. As a comparison, although chordal graphs are sandwich monotone and they can be recognized in linear time, the best known running time is $O(n^{2.376})$ for a minimal chordal completion algorithm [22], and $O(\Delta m)$ for a minimal chordal deletion algorithm [11], where Δ is the largest degree in the input graph.

4 Minimal threshold completions and deletions

A graph $G = (V, E)$ is called a *threshold graph* if there exist nonnegative real numbers w_v for $v \in V$, and t such that for every $I \subseteq V$, $\sum_{v \in I} w_v \leq t$ if and only if I is an independent set [9, 17, 28].

A graph is a *split graph* if its vertex set can be partitioned into a clique and an independent set. The partition of the vertices of a split graph into an independent set S and a clique K is called *split partition* (S, K), and it is not necessarily unique. Split graphs can be recognized and a split partition can be computed in linear time [17]. It is known that a graph is split if and only if it does not contain

any vertex subset inducing $2K_2$, C_4, or C_5 [14]. Hence the next theorem states that a graph is threshold if and only if it is split and does not contain any vertex set inducing a P_4.

Theorem 4.1 ([9]). *A graph is a threshold graph if and only if it does not contain any vertex set inducing $2K_2$, C_4, or P_4.*

Consequently, in a disconnected threshold graph there is at most one connected component that contains an edge. An ordering $v_1, v_2, \ldots, v_{|S|}$ of a subset $S \subseteq V(G)$ of vertices is called *nested neighborhood ordering* if it has the property that $(N_G(v_1) \setminus S) \subseteq (N_G(v_2) \setminus S) \subseteq \ldots \subseteq (N_G(v_{|S|}) \setminus S)$.

Theorem 4.2 ([28]). *A graph is a threshold graph if and only if it is split and the vertices of the independent set have a nested neighborhood ordering in any split partition.*

Threshold graphs can be recognized and a nested neighborhood ordering can be computed in linear time, since sorting the vertices of the independent set by their degrees readily gives such an ordering for threshold graphs [17]. It is NP-hard to compute minimum threshold completions of arbitrary graphs; even split graphs [31].

A split partition of a threshold graph is never unique [17]. Here we define a *threshold partition* (S, K) of a threshold graph in a unique way. Note first that all vertices of degree more than $\omega(G) - 1$ must belong to K, and all vertices of degree less than $\omega(G) - 1$ must belong to S. The set of vertices of degree exactly $\omega(G) - 1$ is either an independent set or a clique [18]. If this set is a clique, we place all of these vertices in K, and if it is an independent set we place them in S. We refine the sets S and K further as follows: $(S_0, S_1, S_2, ..., S_\ell)$ is a partition of S such that S_0 is the set of isolated vertices, and $N(S_1) \subset N(S_2) \subset \ldots \subset N(S_\ell)$, where ℓ is as large as possible. Hence all vertices in S_i have the same degree for $0 \leq i \leq \ell$. This also defines a partition $(K_1, K_2, ..., K_\ell)$ of K, where $K_1 = N(S_1)$ and $K_i = N(S_i) \setminus N(S_{i-1})$ for $2 \leq i \leq \ell$. The remaining vertices of K form another set $K_{\ell+1} = K \setminus N(S_\ell)$. Again, all vertices in K_i have the same degree for $1 \leq i \leq \ell + 1$. By definition, a graph is threshold if and only if its vertex set admits such a threshold partition. Moreover the threshold partition of a threshold graph is unique and all sets S_i and K_i, $1 \leq i \leq \ell$, are non-empty except possibly the sets S_0 and $K_{\ell+1}$. If $K_{\ell+1} = \emptyset$ then S_ℓ contains at least two vertices, and if $K_{\ell+1} \neq \emptyset$ then $K_{\ell+1}$ contains at least two vertices [17].

Lemma 4.3. *Let G be a threshold graph with threshold partition $((S_0, ..., S_\ell), (K_1, ..., K_{\ell+1}))$ and let uv be an edge satisfying the following: either $u \in S_i$ and $v \in K_i$ for some $i \in \{1, ..., \ell\}$, or $u, v \in K_{\ell+1}$. Then $G - uv$ is a threshold graph.*

Proof. Assume first that $u \in S_i$ and $v \in K_i$ for some i. Removing uv from G results in a split graph since we remove and edge between the clique and the independent set of the split partition. In the graph $G' = G - uv$ we have that $N_{G'}(S_{i-1}) \subseteq N_{G'}(u) \subset N_{G'}(S_i \setminus \{u\})$. Thus the new independent set has a nested neighborhood ordering also in G', and hence G' is threshold by Theorem 4.2.

Assume now that $u, v \in K_{\ell+1}$. We describe the threshold partition of G': If $K_{\ell+1}$ contains more than two vertices then the new threshold partition has the sets (K_i, S_i), $1 \leq i \leq \ell$, unchanged, and it also contains the new sets $K'_{\ell+1} = K_{\ell+1} \setminus \{u, v\}$ and $S'_{\ell+1} = \{u, v\}$. If $K_{\ell+1}$ has exactly two vertices, then every set remains as before, except $K_{\ell+1}$ which is removed and S_ℓ which now becomes $S'_\ell = S_\ell \cup \{u, v\}$. It is easy to check that removal of uv results in exactly the described threshold partition for G' and thus G' is a threshold graph. □

For simplicity, we will call an edge uv of G that satisfies the condition in Lemma 4.3 a *candidate edge* of G.

Lemma 4.4. *Let $G = (V, E)$ and $G' = (V, E \cup F)$ be two threshold graphs such that $F \cap E = \emptyset$ and $F \neq \emptyset$. At least one edge in F is a candidate edge of G'.*

Proof. Let $((S'_0, ..., S'_\ell), (K'_1, ..., K'_{\ell+1}))$ be the threshold partition of G'. Assume for a contradiction that F does not contain any candidate edge, and let $uv \in F$. Since uv cannot be a candidate edge, it

is of one of the following three types: (i) $u, v \in K'_i$, for some i satisfying $1 \leq i \leq \ell$, (ii) $u \in K'_i$ and $v \in K'_j$, for some i and j satisfying $1 \leq i < j \leq \ell + 1$, or (iii) $u \in K'_i$ and $v \in S'_j$, for some i and j satisfying $1 \leq i < j \leq \ell$. (Recall that there are no edges uv in G' with $u \in K'_i$ and $v \in S'_j$ where $j < i$ by the definition of threshold partition.)

If uv is of type (i), then since K'_i is non-empty, there is a vertex $x \in S'_i$ such that ux and uv are edges of G'. Since there are no candidate edges in F, $ux, uv \in E$. Assume first that $i \neq \ell$. Then there is a vertex $y \in K'_\ell$ such that uy and vy are edges of G' and xy is not an edge of G'. If both uy and vy belong to E, then $\{u, x, v, y\}$ induces a C_4 in G. If exactly one of uy and vy belongs to E, say uy, and the other belongs to F, then $\{y, u, x, v\}$ induces a P_4 in G. If both uy and vy belong to F, then there is a vertex $z \in S'_l$ such that $\{x, u, z, y\}$ induces a $2K_2$ in G, since zy is a candidate edge of G' and hence cannot be in F. Hence all possibilities lead to a contradiction since G is a threshold graph and cannot contain any of the mentioned induced subgraphs. If $i = \ell$ and $K'_{\ell+1} \neq \emptyset$ then there are at least two vertices y and z in $K'_{\ell+1}$ that can substitute the role of y and z as in the case $i \neq \ell$, since yz is a candidate edge of G' and hence must belong to E. If $i = \ell$ and $K'_{\ell+1} = \emptyset$ then we know that there are at least two vertices $y, z \in S'_\ell$, and that $uy, uz, vy, vz \in E$ (since they are candidate edges). Hence $\{u, y, v, z\}$ induces a C_4 in G, contradicting that G is threshold.

If uv is of type (ii), assume first that $j \neq \ell + 1$. Then we know that there is a vertex $x \in S'_i$ and a vertex $y \in S'_j$ such that $ux, vy \in E$ since they are candidate edges. We know that xv is not an edge of G'. If $yu \in E$ then $\{x, u, y, v\}$ induces a P_4 in G, and if $yu \in F$ then the same set induces a $2K_2$ in G, contradicting in both cases that G is threshold. If $j = \ell + 1$ then we know that there is at least one more vertex $z \neq v$ in $K'_{\ell+1}$ where $vz \in E$ (since it is a candidate edge). If uz belongs to F then $\{v, z, u, x\}$ induces a $2K_2$ in G. Otherwise this set induces a P_4 in G, because zx and vx are not edges in G or G', contradicting that G is threshold.

If uv is of type (iii), then we know that there are vertices $x \in S'_i$ and $y \in K'_j$ such that $ux, vy \in E$ by the same arguments as before. If $uy \in E$ then $\{x, u, y, v\}$ induces a P_4 in G, and if $uy \in F$, then this set induces a $2K_2$ in G. Hence by by Theorem 4.1, we reach a contradiction in all possible cases. Consequently, either G is not threshold, or F must contain a candidate edge, and the proof is complete. □

Theorem 4.5. *Threshold graphs are sandwich monotone.*

Proof. Given G and G' as in the premise of Lemma 4.4, we know by Lemmas 4.3 and 4.4 that there is an edge $f \in F$ such that $G' - f$ is threshold. Now the same argument can be applied to G and $G' = G' - f$ with $F = F \setminus \{f\}$ repeatedly, until G' becomes equal to G. □

4.1 Characterizing and extracting minimal threshold completions and deletions

Theorem 4.6. *Let G be an arbitrary graph and H be a threshold completion of G. H is a minimal threshold completion of G if and only if no fill edge is a candidate edge of H.*

Proof. If H is a minimal threshold completion of G, then there cannot be any fill edge that is a candidate edge of H, because otherwise the removal of this edge would result in a threshold graph by Lemma 4.3, contradicting that H is minimal. If H is not a minimal threshold completion of G, then there exists a minimal threshold completion M of G such that $E(G) \subseteq E(M) \subset E(H)$. By Lemma 4.4 there is an edge $e \in E(H) \setminus E(M)$ that is a candidate edge of H. □

Now given any threshold completion H of an arbitrary graph G, let us consider the problem of extracting a minimal threshold completion H' of G such that $G \subseteq H' \subseteq H$. Algorithm Extr_Min_Threshold describes such a way for computing H'. The basic idea is that we visit vertices of H, remove redundant fill edges incident to them according to Theorem 4.6, and update properly the resulting threshold-partition.

Before proving the correctness of the algorithm it is important to note the following. Vertex v that we choose at each step belongs to either S or $K_{\ell+1}$, since $d_{H'}(v) \leq \omega(H') - 1$. Furthermore we know

Algorithm Extr_Min_Threshold

Input: A graph $G = (V, E)$ and a threshold graph $H = (V, E \cup F)$ with $F \cap E = \emptyset$.
Output: A minimal threshold completion $H' = (V, E \cup F')$ of G such that $F' \subseteq F$.

$H' = H$; Unmark all vertices of H';
Let $(S = (S_0, S_1, ..., S_\ell), K = (K_1, K_2, ..., K_{\ell+1}))$ be the threshold partition of H';
While there is an unmarked vertex v such that $d_{H'}(v) \leq \omega(H') - 1$ **do**
  Pick an unmarked vertex v of minimum $d_{H'}(v)$;
  Find a vertex $u \in N_G(v)$ of minimum $d_{H'}(u)$;
  Compute $U = \{u' \in N_G(v) \mid d_{H'}(u') = d_{H'}(u)\}$;
  Compute the index i for which $U \subseteq K_i$;
  Remove all edges between v and the vertices of $(K_i \setminus U) \cup K_{i+1} \cup \cdots \cup K_j$ from H';
  Update the threshold-partition of H' and mark v;
Return H';

that $U \subseteq K_i$, for some i satisfying $1 \leq i \leq \ell + 1$, since all vertices of U are adjacent to v and have the same degree in H'. Note that all edges that we remove incident to v are fill edges, since we picked u to be the neighbor of v in G with the smallest degree in H', and consequently i is the largest index such that K_i contains a neighbor in G of v.

Lemma 4.7. *Given an arbitrary graph $G = (V, E)$ and a threshold completion $H = (V, E \cup F)$ of G with $F \cap E = \emptyset$, Algorithm Extr_Min_Threshold computes a minimal threshold completion $H' = (V, E \cup F')$ of G, such that $F' \subseteq F$.*

Proof. We have already argued before the statement of the lemma that only fill edges are removed at every step. Hence if H' is a threshold graph then it is a threshold completion of G, and $F' \subseteq F$. For a proof by induction, assume that H' is a threshold graph and that all candidate fill edges are incident to unmarked vertices of degree at most $\omega(H')$, before the step that processes vertex v. This is trivially true before the first step. Let us denote the graph obtained after removing the described edges incident to v by H''. We will show that

- H'' is a threshold graph, and
- if there are any candidate fill edges in H'' then these are incident to unmarked vertices of degree at most $\omega(H'') - 1$.

Consequently, after the last step, since no unmarked vertices of degree at most $\omega(H'') - 1$ will be left in H'', there will be no candidate fill edges in H''. We can then conclude by the above arguments and Theorem 4.6 that the output graph is a minimal threshold completion of G and a subgraph of H.

Let us examine what changes from the threshold partition of H' to the threshold partition of H''. Let $(S' = (S'_0, S'_1, ..., S'_{\ell'}), K' = (K'_1, K'_2, ..., K'_{\ell'+1}))$ be the threshold partition H''. Since v belongs to either S or $K_{\ell+1}$ we distinguish between these two cases.

Case 1. If v belongs to S_j for some j satisfying $1 \leq j \leq \ell$, then $U \subseteq K_i$ for an i satisfying $1 \leq i \leq j \leq \ell$. Note first that if $U = K_j$ then $H'' = H'$. Otherwise we need to remove v from S_j and add it into an appropriate subset of S. Observe also that v is still adjacent in H'' to all vertices of $K_1 \cup K_2 \cup ... \cup K_{i-1} \cup U$. According to how K_i and S_j are related to U and v, respectively, we have the following cases.

If $K_i \neq U$ and $S_j \neq \{v\}$ then $\ell' = \ell + 1$. In particular, $K'_{\ell'+1} = K_{\ell+1}$, $K'_{p+1} = K_p$, and $S'_{p+1} = S_p$ for every $p \in \{i+1, \ldots, \ell\} \setminus \{j\}$. Since v is removed from S_j we have $K'_{j+1} = K_j$ and $S'_{j+1} = S_j \setminus \{v\}$. For the set K_i observe that v now splits it into two parts $K_i \setminus U$ and U. Thus $K'_{i+1} = K_i \setminus U$, $S'_{i+1} = S_i$, $K'_i = U$, and $S'_i = \{v\}$. The rest of the sets remain as before, that is $S'_0 = S_0$, $K'_p = K_p$, and $S'_p = S_p$ for every $p \in \{1, \ldots, i-1\}$.

If $K_i \neq U$ and $S_j = \{v\}$ then $\ell' = \ell$. Now we have $K'_{\ell'+1} = K_{\ell+1}$, $S'_0 = S_0$, $K'_p = K_p$, and $S'_p = S_p$ for every $p \in \{j+2, \ldots, \ell\}$. Since $S_j = \{v\}$ the vertices of K_j are merged with the vertices of K_{j+1}

in H''. Thus we have $K'_{j+1} = K_{j+1} \cup K_j$ and $S'_{j+1} = S_{j+1}$; $K'_p = K_{p-1}$ and $S'_p = S_{p-1}$; for every $p \in \{i+2, \ldots, j\}$; $K'_{i+1} = K_i \setminus U$ and $S'_{i+1} = S_i$; $K'_i = U$ and $S'_i = \{v\}$; and $K'_p = K_p$ and $S'_p = S_p$ for every $p \in \{1, \ldots, i-1\}$.

If $K_i = U$ and $S_j \neq \{v\}$ then $\ell' = \ell$. In this case we obtain $K'_{\ell'+1} = K_{\ell+1}$, $S'_0 = S_0$, $K'_p = K_p$, and $S'_p = S_p$ for every $p \in \{1, \ldots, \ell\} \setminus \{i, j\}$. Since v is removed from S_j we have $K'_j = K_j$ and $S'_j = S_j \setminus \{v\}$. Moreover v is placed into S_i which gives $K'_i = K_i$ and $S'_i = S_i \cup \{v\}$.

If $K_i = U$ and $S_j = \{v\}$ then $\ell' = \ell - 1$. Now we have $K'_{\ell'+1} = K_{\ell+1}$, $S'_0 = S_0$, $K'_{p-1} = K_p$, and $S'_{p-1} = S_p$ for every $p \in \{j+2, \ldots, \ell\}$. Since $S_j = \{v\}$ the vertices of K_j are merged with the vertices of K_{j+1} in H''. Thus we have $K'_j = K_{j+1} \cup K_j$ and $S'_j = S_{j+1}$. The rest of the sets remain as before except S_i. That is, $K'_p = K_p$ and $S'_p = S_p$ for every $p \in \{1, \ldots, j-1\} \setminus \{i\}$, $K'_i = K_i$, and $S'_i = S_i \cup \{v\}$.

Case 2. If $v \in K_{\ell+1}$ then $U \subseteq K_i$, for an i satisfying $1 \leq i \leq \ell + 1$. Observe first that if $U = K_{\ell+1} \setminus \{v\}$ then $H'' = H'$. Otherwise v is removed from $K_{\ell+1}$ and it is placed in the independent set according to whether $K_i = U$ or not. The corresponding cases are analogous to the previous case. That is, either v makes a new set by itself and splits K_i into two parts, or v is placed in a set together with the vertices of S_i. Now we describe what happens to the set $K_{\ell+1}$. Let $K^*_{\ell+1} = K_{\ell+1}$ if $i \neq \ell + 1$ and let $K^*_{\ell+1} = K_{\ell+1} \setminus U$ if $i = \ell + 1$. Recall that $K_{\ell+1}$ contains at least two vertices since it is non-empty. If $|K^*_{\ell+1}| \geq 3$ then v is removed from $K_{\ell+1}$ which gives $K'_{\ell'+1} = K^*_{\ell+1} \setminus \{v\}$. If $K^*_{\ell+1} = \{v, v'\}$ then $K'_{\ell'+1} = \emptyset$. Moreover if $i \neq \ell + 1$ then $K'_{\ell'} \subseteq K_\ell$ and $S'_{\ell'} = S_\ell \cup \{v'\}$; otherwise we have $K'_{\ell'} = U$ and $S'_{\ell'} = \{v, v'\}$.

In all cases we described the threshold partition of H'' which shows that H'' is a threshold graph. Now we will show that if there are candidate edges in H'' then they are incident to unmarked vertices of degree less than $\omega(H'')$.

If $v \in S_j$ (Case 1 above) then v is placed into a set S'_i for which we know that $K'_i = U$. By definition we know that U contains only vertices that are adjacent to v in G. If there are other vertices in S'_i, then these are already marked since their degrees were smaller than that of v before this step. Since $K'_i = K_i$ in such a case, by our induction assumption, we know that there are no candidate fill edges incident to vertices of S'_i. Fill edges that were not candidates in H' might possibly become candidates in H'' only when $S_j = \{v\}$. In that case we have $K'_{j+1} = K_{j+1} \cup K_j$ and $S'_{j+1} = S_{j+1}$. But now we know that S'_{j+1} contains only unmarked vertices, since the degrees in H' of vertices in S'_{j+1} is strictly more than $d_{H'}(v)$. Moreover we know that $K'_{\ell'+1} = K_{\ell+1}$ hence no fill edges inside $K'_{\ell'+1}$ may have become candidates at this step.

Let $v \in K_{\ell+1}$ (Case 2 above). If $|K^*_{\ell+1}| \geq 3$ then $K'_{\ell'+1}$ contains only a subset of $K_{\ell+1}$. Since there are no fill edges in H'' between v and the vertices of U, no candidate fill edges are incident to marked vertices of $K_{\ell+1}$. If $K^*_{\ell+1} = \{v, v'\}$ then v' belongs to $S'_{\ell'}$ and thus it is incident to possible candidate fill edges whose other endpoints are in $K'_{\ell'}$. Now it suffices to show that v' is an unmarked vertex. Observe that vv' is a fill edge. If v' is a marked vertex then it means that for every vertex $x \in K_{\ell+1} \setminus \{v'\}$, $xv' \in E$ since by marking v' and by the definition of U, v' has no fill edges incident to it whose other endpoints are in $K_{\ell+1}$. This contradicts that vv' is a fill edge. Thus by the fact that vv' is a fill edge, $v \in K_{\ell+1}$, and our induction assumption, we can conclude that v' is an unmarked vertex.

Hence at each step of the algorithm we know that every candidate edge of H'' that is incident to an unmarked vertex of degree at most $\omega(H'') - 1$ is an edge of G. □

Lemma 4.8. *The running time of Algorithm Extr_Min_Threshold is $O(n + m + |F|)$.*

Proof. Computing the threshold-partition of H requires scanning the adjacency lists a constant number of times, and sorting the vertices by their degrees, and hence takes a total of $O(n + m + |F|)$ time. For the rest of the operations, we keep a data structure where the degree of v, and the set of the threshold partition that v belongs to is stored in a table and can be checked in constant time, for each vertex $v \in V$. Also, we keep a table showing whether each edge belongs to E or F. After each single fill edge removal, it takes constant time to update the degrees of the endpoints of this edge,

and the degree table. Furthermore, the size of the largest clique and the threshold partition can also be updated in $O(d_H(v))$ time for each visited vertex, by the previous discussion, since at most $d_H(v)$ vertices need to be placed into appropriate sets of the threshold partition. Observe that to find a vertex of H' of minimum degree, we do not need to recompute the degrees of all unprocessed vertices. In the beginning, we can sort all vertices by their degrees in H in a non-decreasing order in $O(n + m + |F|)$ time since largest degree is $O(n)$. When we remove edges incident to a vertex of the independent set, the degrees of the unprocessed vertices in the independent set do not change. Therefore we can continue to pick vertices in the first computed order without having to recompute degrees of or sorting the unmarked vertices. All vertices belonging to S are processed before the vertices of $K_{\ell+1}$, and we can detect when we have started on the vertices of $K_{\ell+1}$ by simply comparing degrees to $\omega(H')$. At this point we can sort the unmarked vertices by their current degrees once more. Finding the set U and the set of fill edges to be deleted for each vertex v requires $O(d_H(v))$ by scanning the adjacency list of v in H' twice, once to find the vertex u in $N_{H'}(v)$ with the smallest degree such that $uv \in E$ and the index i such that $u \in K_i$, and once to delete each fill edge between v and $u' \in N_{H'}(v)$ for vertices u' belonging to $K_i \cup K_{i+1}, ..., K_j$. Thus the overall running time is linear in the size of H. □

With Algorithm Extr_Min_Threshold and the above two lemmas, we have thus proved the following.

Theorem 4.9. *Let $H = (V, E \cup F)$ be a threshold completion of an arbitrary graph $G = (V, E)$ with $F \cap E = \emptyset$. A minimal threshold completion $H' = (V, E \cup F')$ of G, such that $F' \subseteq F$, can be computed in $O(n + m + |F|)$ time.*

Next we proceed to characterizing minimal threshold deletions. Since threshold graphs are both self-complementary and sandwich monotone, H is a minimal threshold deletion of an arbitrary graph G if and only if $\overline{H}$ is a minimal threshold completion of $\overline{G}$. Therefore, first we relate the threshold partition of $\overline{H}$ to the threshold partition of H, for a given threshold graph H and its complement $\overline{H}$. Let $((S_0, ..., S_\ell), (K_1, ..., K_{\ell+1}))$ be the threshold partition of H, and let $((S'_0, ..., S'_{\ell'}), (K_1, ..., K_{\ell'+1}))$ be the threshold partition of $\overline{H}$. There are the following possibilities, that can be verified easily:

- $S_0 \neq \emptyset$: In this case, all vertices of S_0 are universal in $\overline{H}$ and hence $K'_1 = S_0$. Since $S_0 \neq \emptyset$, the vertices of K_1 are not universal in H, and therefore there are no isolated vertices in $\overline{H}$ and $S'_0 = \emptyset$. For $1 \leq i \leq \ell$, the threshold partition of $\overline{H}$ is given by $S'_i = K_i$ and $K'_{i+1} = S_i$. If $K_{\ell+1} = \emptyset$ we are done. If not, $\ell' = \ell + 1$, and $S'_{\ell'} = K_{\ell+1}$ and $K'_{\ell'+1} = \emptyset$.

- $S_0 = \emptyset$: In this case, vertices of K_1 are universal in H and hence they become isolated in $\overline{H}$, giving $S'_0 = K_1$. For $1 \leq i \leq \ell$, we get $S'_{i-1} = K_i$ and $K'_i = S_i$. If $K_{\ell+1} = \emptyset$, we are done and $\ell' = \ell - 1$. Otherwise, $\ell' = \ell$, and $S'_{\ell'} = K_{\ell+1}$ and $K'_{\ell'+1} = \emptyset$.

Theorem 4.10. *Let $G = (V, E)$ be an arbitrary graph and let H be a threshold deletion of G with threshold partition $((S_0, ..., S_\ell), (K_1, ..., K_{\ell+1}))$. H is a minimal threshold deletion of G if and only if there is no deleted edge with the following property: $u \in S_{i-1}$ and $v \in K_i$ for some i satisfying $1 \leq i \leq \ell + 1$, or $u, v \in S_\ell$ for $K_{\ell+1} = \emptyset$.*

Proof. By Theorem 4.6 and the above discussion, we conclude that H is a minimal threshold deletion if and only if no deleted edge of H is a candidate fill edge of $\overline{H}$. Let the threshold partition $((S'_0, ..., S'_{\ell'}), (K'_1, ..., K'_{\ell'+1}))$ of $\overline{H}$ be given as above. Recall that a fill edge uv of $\overline{H}$ is candidate if either $u \in S'_{i'}$ and $v \in K'_{i'}$ for some $i' \in \{1, ..., \ell'\}$, or $u, v \in K'_{\ell'+1}$. By the above discussion, $u \in S'_{i'}$ and $v \in K'_{i'}$ if and only if either $u \in S_{i'-1}$ and $v \in K_{i'}$, or $u \in S_{i'}$ and $v \in K_{i'+1}$. In both cases there is an index i such that $u \in S_{i-1}$ $v \in K_i$. For the second condition, observe first that $K'_{\ell'+1} \neq \emptyset$ if and only if $K_{\ell+1} = \emptyset$. Hence $u, v \in K'_{\ell'+1}$ if and only if $u, v \in S_\ell$ and $K_{\ell+1} = \emptyset$, by the above discussion. □

Theorem 4.11. *Let $H = (V, E \setminus D)$ be a threshold deletion of an arbitrary graph $G = (V, E)$ with $D \subseteq E$. A minimal threshold deletion $H' = (V, E \setminus D')$ of G, such that $D' \subseteq D$ can be computed in $O(n + m)$ time.*

Proof. We describe such an algorithm. At start $H' = H$. Let $(S = (S_0, ..., S_\ell), K = (K_1, ..., K_{\ell+1}))$ be the threshold partition of H' at the start of a step. At each step, we pick an unmarked vertex of largest degree in H' such that $d_{H'}(v) \geq \omega(H') - 1$. Thus we will process all vertices of K, and all vertices of S_ℓ if $K_{\ell+1} = \emptyset$. For the chosen vertex v of a step, we find a non-edge vu of G such that u has maximum degree in H'. Then we compute the set U of non-neighbors in G of v that have the same degree as u in H'. Let $v \in K_j$ and let $U \subseteq S_i$, for some i and j with $0 \leq i < j$. Note that if $v \in S_\ell$ then $U \subseteq S_i$, for some i with $0 \leq i \leq \ell$. Since $u \in U$ has the largest degree in H' among all non-neighbors in G of v, we know that every non-edge in H' between v and vertices of $(S_i \setminus U) \cup S_{i+1} \cup \ldots \cup S_{j-1}$ is a deleted edge and belongs to D. If $v \in S_\ell$ then we know that non-edges of H' between v and the vertices of $(S_i \setminus U) \cup S_{i+1} \cup \ldots \cup S_\ell \setminus \{v\}$ are deleted edges. In both cases we add those edges to H'. At the end of a step, we mark vertex v, and proceed with the next unmarked vertex of largest degree.

To prove the correctness of this algorithm we use the fact it is equivalent to applying Algorithm Extr_Min_Threshold on $\overline{H}$ and $\overline{G}$. First observe that at each step we pick a vertex v of largest degree in H' such that $d_{H'}(v) \geq \omega(H) - 1$. By the discussion above this means that in $\overline{H'}$ we pick the vertex v of smallest degree such that $d_{\overline{H'}}(v) \leq \omega(\overline{H'}) - 1$. The set U of non-adjacent vertices to v in G corresponds to the set of adjacent vertices to v in $\overline{G}$ with the smallest degree in $\overline{H'}$. Adding the deleted edges incident to v is equivalent to removing the corresponding fill edges in $\overline{H'}$ just as the Algorithm Extr_Min_Threshold does. Therefore the proposed algorithm computes a minimal threshold deletion H' of G since by Theorem 4.9 Algorithm Extr_Min_Threshold computes a minimal threshold completion $\overline{H'}$ of $\overline{G}$, and H' is a minimal threshold deletion of G if and only if $\overline{H'}$ is a minimal threshold completion of $\overline{G}$.

The reason why we give this algorithm directly on G and H rather than just applying Algorithm Extr_Min_Threshold on the complement graphs is because of the running time. Since $E(\overline{G})$ might be $O(n^2)$, applying the previous algorithm directly on the complements would not necessarily result in running time which is linear in the size of G. To obtain the linear running time, like in the completion case, we start by computing a threshold partition of H in $O(n + m)$ time. The degree of each vertex v, and the set of the threshold partition that v belongs to is stored in a table, whereas the knowledge about whether an edge of E belongs to D is maintained in another table. Then we sort the vertices by their degrees in H in a non-increasing order. Again, since we only add edges incident to the chosen vertex, the degrees of the unmarked vertices of K do not change, and we can process all vertices in the originally sorted order. By using $\omega(H)$ we know when we reach the vertices of S_ℓ. At that point we sort the vertices again as we do with completion case. In order to find the set of vertices U we need to spend $O(d_G(v))$ time for each vertex v that we visit. For that purpose first we scan $N_G(v)$ and count how many neighbors of v are inside a set S_i, for each index i satisfying $i < j$ if $v \in K_j$, and $i \leq \ell$ if $v \in S_\ell$. Then we compare these numbers with the size of each S_i, and find in this way the largest index $i < j$ (or $i \leq \ell$) such that S_i contains a non-neighbor of in G of v. Thus in $O(d_G(v))$ time we know that between v and a vertex of S_i there is a non-edge of G, whereas between v and the vertices of $S_{i+1}, \ldots, S_{j-1}$ there are only deleted edges. If $v \in S_\ell$ then there are only deleted edges between v and the vertices of $S_{i+1}, \ldots, S_\ell$. Notice also that if $i = j - 1$ then $S_i = S_{j-1}$ if $v \in K_j$, and $S_i = S_\ell$ if $v \in S_\ell$. By scanning again $N_G(v)$ we find the correct set $U \subseteq S_i$. Adding the deleted edges to H' takes the amount of $O(d_G(v))$ time in order to update the threshold partition of H', since for at most $d_G(v)$ vertices we need to update their sets in the threshold partition, as explained in the completion case. Therefore summing up the time needed for each vertex we get the overall linear time complexity of the algorithm. □

Corollary 4.12. *Any minimal threshold deletion of an arbitrary graph can be computed in $O(n + m)$ time.*

Proof. Let $G = (V, E)$ be the input graph, and let $D = E$. The graph $H = (V, E \setminus D)$ is a threshold graph since edgeless graphs are threshold, and hence H is a threshold deletion of G. By Theorem 4.11 a minimal threshold deletion H' of G can be computed in $O(n + m)$ time. Since $D = E$, we have the possibility of choosing any subset of the edges that will give a minimal deletion of G. □

4.2 Computing a minimal threshold completion directly

Here we show how to obtain a minimal threshold completion H of G directly. The motivation for this is that we now compute H in time linear in the size of G. The idea behind our approach is the following: Compute a minimal split completion of G using the algorithm of [18], and then compute a minimal threshold completion of the computed split completion by giving the vertices of the independent set a nested neighborhood ordering. We show that this indeed results in a minimal threshold completion of G.

Theorem 4.13. *A minimal threshold completion of an arbitrary graph G can be computed in $O(n+m)$ time.*

Proof. First we compute a minimal split completion G' of G in $O(n+m)$ time by the algorithm given in [18]. The output of this algorithm does not list all edges of G'; it lists all edges of G, gives a split partition (K, S) of G', and the knowledge that all fill edges have both endpoints in K. In case there are any isolated vertices in S, we define a new set S' which are all the non-isolated vertices of S. Now we compute an order of the vertices of S' such that $d(v_1) \geq d(v_2) \geq \ldots \geq d(v_{|S'|})$. The important point is that $d_{G'}(v) = d_G(v)$ for each $v \in S$, hence this can be done in $O(n+m)$ time. For each vertex v_i we make it adjacent to the vertices of $N(\{v_{i+1}, v_{i+2}, \ldots, v_{|S'|}\})$, starting from v_1. That is, at each step, we pick an unmarked vertex of the independent set with the largest degree and add edges to the neighborhood of the unmarked vertices of the independent set except the isolated vertices. At the end, the resulting graph remains a split graph since we only add edges between S and K, and $v_1, v_2, ..., v_{|S'|}, v_{|S'|+1}, ..., v_{|S|}$ is a nested neighborhood ordering. Hence the resulting graph is threshold by Theorem 4.2.

Regarding minimality note the following. The algorithm for the minimal split completion adds fill edges only within the set K [18]. Given the minimal split completion G' of G computed by this algorithm, the algorithm described above adds fill edges only between a vertex of S and a vertex of K. The removal of any single fill edge of the split algorithm gives a non-split graph by the results of [18]. For an edge that we added to the split graph G', say v_iu_j, where $v_i \in S$ and $u_j \in K$ with $i < j$, we know that u_j must be a neighbor of another vertex v_j, where $v_j \in S$, and the edge u_jv_j is an edge of G because the minimal split completion does not add edges between S and K. Then there is another vertex u_i such that $u_i \in N_{G'}(v_i)$ and $u_i \notin N_{G'}(v_j)$, since $d_{G'}(v_i) \geq d_{G'}(v_j)$. Note that if $N_{G'}(v_i) = N_{G'}(v_j)$ then v_iu_j is not a fill edge. Thus the removal of a fill edge v_iu_j results the $P_4 = v_iu_iu_jv_j$ in the graph $G'' = G' - v_iu_j$ which implies that G'' is a non-threshold graph by Theorem 4.1. Hence no single fill edge can be removed, since either a non-split graph is obtained or the nested neighborhood ordering is destroyed. Therefore, by Theorem 4.5, the algorithm that we described for computing a minimal threshold completion is correct.

To complete the argument about the running time, for each vertex $v_i \in S$, starting from v_1, we give its neighbors, which lie in the clique, the label i, $1 \leq i \leq |S'|$. At the end, each vertex of the clique of G' with label k knows that it must be adjacent to every vertex from v_1 to v_k given in the ordering of S'. Now adding these edges between the clique and the independent set according to the label of each vertex of K, we get a linear time bound in the size of the *output* graph. To bound the running time to be linear in the size of the *input* graph, instead of representing the output graph explicitly (for example, given by its adjacency list), we output a unique $O(n)$ space representation of it: We output both the sequence $v_1, v_2, ..., v_{|S|}$, and for every vertex of the clique its label. Hence, we can skip the step of adding edges between K and S since the ordering and the resulting labels of the vertices of K, together with the input graph, uniquely define the edges of the threshold completion. Therefore all steps can be done in total $O(n+m)$ time. □

5 Minimal bipartite deletions and co-bipartite completions

A graph is *bipartite* if its vertex set can be partitioned into two independent sets. The partition of a bipartite graph G into two independent vertex sets is called *bipartition* and this partition is unique if

and only if G is connected. Bipartite graphs are exactly the class of graphs that do not contain cycles of odd length [27]. It is well known that simple modifications of breadth-first search (BFS) can be used to find an odd cycle in a graph or provide a bipartition of it in linear time. The complement of a bipartite graph is called *co-bipartite graph* and the bipartition into two independent sets of a bipartite graph is a bipartition into two cliques in its complement.

Since every graph can be deleted into a bipartite graph, and every graph can be completed into a co-bipartite graph, but bipartite completions and co-bipartite deletions are not meaningful for all graphs, we study only minimal bipartite deletions and minimal co-bipartite completions. We should also mention that computing minimum bipartite deletions and minimum co-bipartite completions are NP-hard problems [15].

5.1 Minimal bipartite deletions

A minimal deletion is equivalent to a maximal spanning subgraph. If the input graph G is connected then there exists a connected bipartite deletion of G, since G has a spanning tree, and trees are bipartite. If G is not connected, then a minimal bipartite deletion of G is a collection of connected minimal bipartite deletions of the connected components of G. For this reason, we give results for connected input graphs here, and if G is disconnected then the results can be applied to each connected component of G separately. Since bipartite graphs are monotone, they are trivially also sandwich monotone. First we characterize minimal bipartite deletions.

Lemma 5.1. *Let G be an arbitrary connected graph and let H be a bipartite deletion of G with bipartition (X, Y). H is a minimal bipartite deletion of G if and only if H is connected and no deleted edge has one endpoint in X and one endpoint in Y.*

Proof. Since bipartite graphs are sandwich monotone, by Observation 3.2 H is minimal if and only if no single deleted edge can be added back without destroying bipartiteness. Let H be a minimal bipartite deletion of G. If H is not connected, then any single deleted edge of G with endpoints in two different connected components of H can be added back without destroying bipartiteness, which gives a contradiction to the minimality of H. Hence H is connected, and therefore its bipartition (X, Y) is unique (upon exchanging X and Y). If there is any deleted edge that has one endpoint in X and the other in Y, then this deleted edge can be added back without destroying bipartiteness, which again contradicts the minimality of H. Hence, no such deleted edge can exist if H is minimal. For the other direction, assume that H is connected and all deleted edges have both their endpoints either within X or within Y. Since H is connected, all paths between pairs of vertices in X contain an odd number of vertices, and the same is true for Y. Hence no single edge can be added between endpoints that both belong to X or both belong to Y, since this will introduce an odd cycle and destroy bipartiteness. Since no single deleted edge can be added back, H is minimal. □

Theorem 5.2. *Let $H = (V, E \setminus D)$ be a bipartite deletion of an arbitrary graph $G = (V, E)$ with $D \subseteq E$. A minimal bipartite deletion $H' = (V, E \setminus D')$ of G, such that $D' \subseteq D$ can be computed in $O(n + m)$ time.*

Proof. Assume that G is connected (otherwise we compute the connected components of G is $O(n+m)$ time, and apply the following algorithm to each of them in a total of $O(n+m)$ time). By Lemma 5.1 we know that H' must be connected. If H is disconnected, since G is connected, there are edges in D that can be added to H to obtain a connected bipartite graph H''. This can be done by scanning all edges of D and adding them to H as long as no cycles are created, in a similar way as computing spanning trees and hence in $O(n + m)$ time. Now, the obtained graph H'' is connected and it has a unique bipartition (X, Y). This bipartition can be computed and all vertices can be marked with the set they belong to in $O(n+m)$ time. Finally, we add all remaining edges of D with one endpoint in X and one endpoint in Y to H'' to obtain the desired bipartite deletion H'. This takes $O(|D|) = O(m)$ time. Let D' be the set of all edges of D that are not added back during this process. Clearly, $H' = (V, E \setminus D')$

and all edges of D' have both endpoints in X or both endpoints in Y. Hence by Lemma 5.1, H' is a minimal bipartite deletion of G. □

Corollary 5.3. *Any minimal bipartite deletion of an arbitrary graph can be computed in $O(n+m)$ time.*

Proof. Assume without loss of generality that $G = (V, E)$ is an arbitrary connected graph. Let $D = E$. Clearly $H = (V, E \setminus D)$ is a bipartite deletion of G. Apply the algorithm in the proof of Theorem 5.2 to find a minimal bipartite deletion of G in linear time. Since $D = E$ we have the possibility of adding back any subset of the edges that gives a minimal deletion. □

5.2 Minimal co-bipartite completions

By Observation 3.2 co-bipartite graphs are sandwich monotone. Co-bipartite graphs can be recognized in linear time, since BFS can be applied in the complement of a graph in time linear in the size of the input graph [8, 25]. A co-bipartition of a co-bipartite graph G is unique if and only if $\overline{G}$ is connected because of the unique bipartition of $\overline{G}$. First we characterize minimal co-bipartite completions.

Lemma 5.4. *Let G be an arbitrary graph and let $V_1, V_2, \ldots, V_k$ be the vertex sets of the connected components of $\overline{G}$. Let H be a co-bipartite completion of G and let (A, B) be a co-bipartition of H. H is a minimal co-bipartite completion of G if and only if $\overline{H}[V_i]$ is connected for $1 \leq i \leq k$, and no fill edge has one endpoint in A and one endpoint in B.*

Proof. Analogous to the discussion about threshold completions and deletions, observe that H is a minimal co-bipartite completion of G if and only if $\overline{H}$ is a minimal bipartite deletion of $\overline{G}$, since adding edges to G is equivalent to removing edges from $\overline{G}$, and H is co-bipartite if and only if $\overline{H}$ is bipartite. For each V_i, $1 \leq i \leq k$, let (A_i, B_i) be a bipartition of $\overline{H}[V_i]$. Consequently, by Lemma 5.1, $H[V_i]$ is a minimal co-bipartite completion of $G[V_i]$ if and only if $\overline{H}[V_i]$ is connected and no fill edge of $H[V_i]$ has one endpoint in A_i and one endpoint in B_i, for $1 \leq i \leq k$. To complete the proof, observe there are no fill edges in H with an endpoint in A_i and an endpoint in A_j or B_j with $i \neq j$, because there are no edges between V_i and V_j in $\overline{G}$, which means that all edges between these two sets are present in G. Thus, no matter how the co-bipartition (A, B) of H is chosen, there are no fill edges between the two sides of the co-bipartition. □

Theorem 5.5. *Let $H = (V, E \cup F)$ be a co-bipartite completion of an arbitrary graph $G = (V, E)$. A minimal co-bipartite completion $H' = (V, E \cup F')$ of G, such that $F' \subseteq F$ can be computed in time $O(n + m + |F|)$.*

Proof. Observe first that $n + m + |F| = \Theta(n^2)$ because at least one side of the co-bipartition must contain $\Theta(n^2)$ edges in a co-bipartite graph. Hence $|E(\overline{H})| = O(m + |F|)$. Therefore, we can compute $\overline{G}$ and $\overline{H}$ in $O(n + m + |F|)$ time. By the discussion in the proof of Lemma 5.4, $\overline{H}$ is a bipartite deletion of $\overline{G}$. By Theorem 5.2 we can compute a minimal bipartite deletion $\overline{H'}$ of $\overline{G}$, where $\overline{H'}$ is a supergraph of $\overline{H}$, in $O(n + m + |F|)$ time. By the above discussion, H' is then a minimal co-bipartite completion of G, and H' is a subgraph of H. □

Now we give an algorithm for computing a co-bipartite completion of an input graph directly in time which is linear in the size of the input graph.

Theorem 5.6. *A minimal co-bipartite completion of an arbitrary graph can be computed in $O(n+m)$ time.*

Proof. Let G be the input graph. Compute the vertex sets $V_1, V_2, \ldots, V_k$ of the connected components of $\overline{G}$. For each connected graph $\overline{G}[V_i]$, compute a BFS tree T_i of it. Since T_i is bipartite and connected, we have a unique bipartition (A_i, B_i) of T_i. Now a co-bipartition (A, B) for the output graph can be chosen by placing A_i and B_i on opposite sides of the co-bipartition for each i. We define the output

graph H as follows: Add to G the missing edges so that A becomes a clique and B becomes a clique. Clearly H is a co-bipartite completion of G. Furthermore, no fill edge of H is between A and B, and thus H is a minimal co-bipartite completion of G by Lemma 5.4 if $\overline{H}[V_i]$ is connected for each i. To see that $\overline{H}[V_i]$ is connected, observe that every edge of T_i is between A_i and B_i, and since we have not added any fill edges between A_i and B_i, the non-edges of G between these two sets remain non-edges in H and hence edges in $\overline{H}$. Therefore, since each T_i is connected, each $\overline{H}[V_i]$ is also connected.

For the running time, note that computing the connected components of $\overline{G}$ and a BFS on $\overline{G}$ can be performed in $O(n+m)$ time by the results of [8, 25]. Moreover instead of adding the fill edges explicitly to H, which is time consuming, we just output the co-bipartition (A, B). Since no fill edges are added between A and B, this together with the input graph defines the output graph uniquely. The total $O(n+m)$ running time follows. □

6 Minimal chain completions and deletions

Yannakakis introduced *chain graphs* and defined a bipartite graph to be a chain graph if one of the sides of the bipartition has nested neighborhood ordering [38]. He also showed that one side has this property if and only if both sides have the property. Chain graphs can be recognized in linear time [28]. It is also known that a graph is a chain graph if and only if it does not contain a vertex set inducing $2K_2$, C_3, or C_5 as an induced subgraph [28]. Computing a minimum chain completion or a minimum chain deletion of a bipartite graph is an NP-hard problem [37].

Theorem 6.1 ([28]). *A graph G is a chain graph if and only if G is bipartite and turning one side of the bipartition into a clique gives a threshold graph for any bipartition of G.*

By Theorem 6.1, a chain graph G can have at most one connected component that contains an edge. Isolated vertices can belong to any side of the bipartition. We will here define a unique way of partitioning the vertices of a chain graph that we call *chain partition*, similar to threshold partition. Define X_0 to be the set of all isolated vertices of G. The remaining vertices induce a connected bipartite graph which thus has a unique bipartition (X, Y). Partition X into $(X_1, X_2, ..., X_\ell)$ where $N_G(X_1) \subset N_G(X_2) \subset \ldots \subset N_G(X_\ell)$, and ℓ is as large as possible. Hence vertices of X_i have the same neighborhood, for each i. This defines a partition of Y into $(Y_1, Y_2, ..., Y_\ell)$, where $Y_i = N_G(X_i) \setminus N_G(X_{i-1})$, $2 \leq i \leq \ell$. Observe that each set X_i contains at least one vertex which implies that the set Y_i is also a non-empty set. If there are only isolated vertices in G, we let $\ell = 0$. The chain partition is unique (upon exchanging X with Y).

Theorem 6.2. *Chain graphs are sandwich monotone.*

Proof. Let $G = (V, E)$ and $G' = (V, E \cup F)$ be two chain graphs such that $E \cap F = \emptyset$ and $F \neq \emptyset$. We will show that there is an edge $f \in F$ that can be removed from G' so that the resulting graph remains a chain graph, from which the result follows by induction on the edges in F.

Let $((X'_0, X'_1, ..., X'_\ell), (Y'_1, Y'_2, ..., Y'_\ell))$ be the chain-partition of G'. First we prove that if F contains an edge xy such that $x \in X'_i$ and $y \in Y'_i$ for some $i \in \{1, ..., \ell\}$ then removing xy from G' results in a chain graph. Let G'' be the graph that we obtain after removing xy; that is, $G'' = G' - xy$. In G'' we have that $N_{G''}(X'_{i-1}) \subset N_{G''}(x) \subset N_{G''}(X'_i \setminus \{x\}) \subset N_{G''}(X'_{i+1})$. Thus the nested neighborhood ordering is maintained which implies that G'' is a chain graph.

Now we prove that F contains at least one edge with one endpoint in X'_i and one endpoint in Y'_i, for some i satisfying $1 \leq i \leq \ell$. Assume for a contradiction that there are no edges in F with one endpoint in X'_i and the other in Y'_i. Hence for every edge $xy \in F$, $x \in X'_j$ and $y \in Y'_i$, where $1 \leq i < j \leq \ell$. Since X'_j and Y'_i are non-empty, X'_i and Y'_j are non-empty, too by definition. Let $x' \in X'_i$ and $y' \in Y'_j$. Edges xy' and $x'y$ cannot belong to F and hence they are edges in E by our assumption. Edge $x'y'$ is not an edge of G' by the definition of chain partition, and hence it is not an edge of G either. Since xy is not an edge of G, $\{x, y', x', y\}$ induces a $2K_2$ in G contradicting that G is a chain graph. □

6.1 Characterizing and extracting minimal chain completions and deletions

Speaking about chain completions of arbitrary graphs is only meaningful if every graph can be embedded in a chain graph by adding edges. Thus it is reasonable to restrict for chain completions the input graph to be a bipartite graph. However any graph can be turned into chain by deleting edges, thus, the input for chain deletion is an arbitrary graph.

Lemma 6.3. *Let G be a bipartite graph, and let H be a chain completion of G with chain partition $((X_0, X_1, ..., X_\ell), (Y_1, Y_2, ..., Y_\ell))$. H is a minimal chain completion of G if and only if no fill edge has one endpoint in X_i and and one endpoint in Y_i for some $i \in \{1, ..., \ell\}$.*

Proof. By sandwich monotonicity H is a minimal chain completion of G if and only if the removal of any single fill edge from H destroys the chain property. Assume that H is a minimal chain completion of G. Any fill edge with one endpoint in X_i and one endpoint in Y_i can be removed without destroying the chain property as we showed in the proof of Theorem 6.2, and hence there cannot be any such fill edges since H is minimal. If H is not minimal, then we know that there is a minimal chain completion M of G that is a subgraph of H such that $E(G) \subseteq E(M) \subset E(H)$. Hence by the proof of Theorem 6.2, there is an edge $e \in E(H) \setminus E(M)$, with one endpoint X_i and one endpoint in Y_i for some $i \in \{1, ..., \ell\}$. This edge e is a fill edge, and the proof is complete. □

Now let us consider the problem of extracting a minimal chain completion H' from any chain completion H of an arbitrary bipartite graph G.

Theorem 6.4. *Let $H = (V, E \cup F)$ be a chain completion of a bipartite graph $G = (V, E)$. A minimal chain completion $H' = (V, E \cup F')$ of G, such that $F' \subseteq F$ can be computed in time $O(n + m + |F|)$.*

Proof. Let $(X = (X_0, X_1, ..., X_\ell), Y = (Y_1, ..., Y_\ell))$ be the chain partition of H. Add edges between all pairs of vertices in Y to obtain a graph H_t. By Theorem 6.1, H_t is a threshold graph and hence a threshold completion of G. The chain partition of H is the threshold partition of H_t, since all vertices of Y have neighbors in X. Let us run Algorithm Extr_Min_Threshold with input G and H_t, and run it as long as there are unmarked vertices in X, and stop after the last vertex of X is processed. This modified algorithm will output a threshold completion H'_t of G such that H'_t is a subgraph of H_t, and only fill edges between X and Y are removed. Let us remove all edges with both endpoints in Y from H'_t to obtain a bipartite graph H'. Clearly H' is a chain completion of G, and H' is a subgraph of H. By the proof of Lemma 4.7, H' is a minimal chain completion of G.

We need to argue that this can be done in time linear in the size of H. To do this, we do not actually add all edges between the vertices of Y before running the mentioned modified algorithm. Since we will only process the vertices of X, we do not need to compare their degrees to the largest clique in H_t. So we simply sort all vertices of X by their degrees in H, and then process them in the sorted order, exactly in the same way as in Algorithm Extr_Min_Threshold, until the last vertex of X is processed. This takes $O(n+m+|F|)$ time, since these steps only consider the edges between X and Y. □

Next we consider chain deletions.

Lemma 6.5. *Let $G = (V, E)$ be an arbitrary graph, and let H be a chain deletion of G with chain partition $(X = (X_0, X_1, ..., X_\ell)$, $Y = (Y_1, ..., Y_\ell))$. H is a minimal chain deletion of G if and only if no deleted edge uv has the following property: $u \in X_{i-1}$ and $v \in Y_i$ for some $1 \leq i \leq \ell$, or $u \in X_\ell$ and $v \in X_0$.*

Proof. Assume that a deleted edge uv has the described property. Let H' be the graph obtained by adding uv to H. If $u \in X_{i-1}$ and $v \in Y_i$ for some $1 \leq i \leq \ell$, then H' remains bipartite and its bipartition is (X, Y). Moreover X has a nested neighborhood ordering in H' since $N_{H'}(X_{i-1} \setminus \{u\}) \subset N_{H'}(u) \subseteq N_{H'}(X_i)$. If $u \in X_\ell$ and $v \in X_0$ then $(X \setminus \{v\}, Y \cup \{v\})$ is a bipartition of H' and H' admits the same chain partition as that of H except the sets X_0 and Y_ℓ which become $X_0 \setminus \{v\}$ and $Y_\ell \cup \{v\}$,

respectively. Hence H' is a chain graph, and therefore H is not a minimal chain deletion of G. We have thus proved that if H is minimal then there cannot be any deleted edges as described.

For the opposite direction, assume that none of the deleted edges satisfies the mentioned property. We prove that the addition of any deleted edge results in a non-chain graph in this case, from which we can conclude that H is minimal by the sandwich monotonicity of chain graphs. Let uv be a deleted edge, and let H' be the graph obtained by adding uv to H. Then uv is one of the following three types: (i) either $u, v \in Y$ or $u, v \in X \setminus X_0$, (ii) $u \in X_{i-1}$ and $v \in Y_j$, for some $1 \leq i < j \leq \ell$, (iii) either $u \in X_0$ and $v \in X \setminus (X_0 \cup X_\ell)$, or $u, v \in X_0$ and $\ell \geq 1$. If uv is of type (i) then H' is not bipartite since $H - X_0$ is a connected bipartite graph with a unique bipartition and both endpoints of uv belong to the same side of bipartition which gives an odd cycle in H'. If uv is of type (ii) then there exist $x \in X_i$ and $y \in Y_i$, such that $\{x, y, u, v\}$ induces a $2K_2$ in H', since x is not adjacent to v and y is not adjacent to u, and hence H' is not chain. If uv is of type (iii) then in both subcases we again obtain a $2K_2$ in H' induced by $\{x, y, u, v\}$, where $x \in X_\ell$ and $y \in Y_\ell$, since y is not adjacent to v. □

Next we consider the problem of extracting a minimal chain deletion from any chain deletion H of an arbitrary graph G.

Theorem 6.6. *Let $H = (V, E \setminus D)$ be a chain deletion of an arbitrary graph $G = (V, E)$ with $D \subseteq E$. A minimal chain deletion $H' = (V, E \setminus D')$ of G, such that $D' \subseteq D$ can be computed in time $O(n+m)$.*

Proof. We describe a similar algorithm to the one given in the proof of Theorem 4.11. Initially, we unmark all vertices and set $H' = H$. Let $(X = (X_0, X_1, ..., X_\ell), Y = (Y_1, ..., Y_\ell))$ be the chain partition of H'. At each step we pick an unmarked vertex $v \in X$ of largest degree in H'. Thus we start by processing the vertices of X_ℓ. If $v \in X_\ell$ then we find the non-edge vu of G such that $u \in X_0$. Otherwise, if $v \in X_i$ for some $0 \leq i < \ell$, then we find the non-edge vu of G such that $u \in Y$ and u has the largest degree in H'. Since H' is a subgraph of G, there is an index $j > i$ such that $u \in Y_j$. Then we compute the set U of the non-neighbors of v in G that have the same degree as u in H'. That is, either $U \subseteq X_0$ or $U \subseteq Y_j$. Observe also that every vertex u of U has the largest degree in H' among non-neighbors of v in G. Thus all edges either between v and vertices of $X_0 \setminus U$, or between v and vertices of $Y_{i+1}, \ldots, Y_{j-1}, Y_j \setminus U$ are deleted edges. In both cases we add those edges to H'. At the end of the step we mark vertex v and proceed with the next unmarked vertex of X of largest degree in H'.

Let us describe what happens after adding the corresponding edges incident to v. Assume that H' is a chain deletion of G and a subgraph of H at the beginning of a step, and let H'' be the graph obtained at the end of a step. We will show that H'' is a chain graph by describing its chain partition $(X' = (X'_0, X'_1, ..., X'_{\ell'}),\ Y' = (Y'_1, ..., Y'_\ell))$. If $v \in X_\ell$ then we add edges between v and vertices of $X_0 \setminus U$. In that case we have $X'_0 = U$ and we consider the set $X_\ell \setminus \{v\}$. If $X_\ell \neq \{v\}$ then we obtain $X'_{\ell+1} = \{v\}$, $Y'_{\ell+1} = X_0 \setminus U$, and $X'_\ell = X_\ell \setminus \{v\}$. Otherwise, we obtain $Y'_\ell = Y_\ell \cup X_0 \setminus U$. In both cases the rest of the sets remain as before. If $v \in X_i$ for some $0 \leq i < \ell$, then we add edges between v and the vertices of $Y_{i+1}, \ldots, Y_{j-1}, Y_j \setminus U$, for an index j satisfying $i < j \leq \ell$. If $Y_j \neq U$ then $Y'_j = Y_j \setminus U$, $Y'_{j+1} = U$, $X'_j = \{v\}$, and $X'_{j+1} = X_j$, whereas if $Y_j = U$ then $X'_{j-1} = X_{j-1} \cup \{v\}$. Moreover in both cases we need to describe the sets X_i and Y_i. If $X_i \neq \{v\}$ then $X'_i = X_i \setminus \{v\}$. Otherwise, $X'_i = X_{i-1}$ and $Y'_i = Y_i \cup Y_{i-1}$. In each case the non-mentioned sets of the chain partition remain as before.

Having described the chain partition of H'', we know that at each step H' remains a chain graph, and hence it is a chain deletion of G at the end of the algorithm. Regarding minimality, analogous to the induction proof of Lemma 4.7, we will prove that if there are deleted edges in H'' with the properties described in Lemma 6.5 then they are incident to unmarked vertices of degree less than $d_H(v)$.

If $v \in X_\ell$ then after adding the corresponding edges no deleted edge is incident to v and a vertex of $X'_0 = U$. Moreover the rest of the marked vertices, if any, lie in X_ℓ, since they have degree at least $d_H(v)$. For those vertices observe that no deleted edge has one endpoint to them and the other to X_0, since they have been visited by the algorithm. Thus there is no deleted edge between $X_\ell \setminus \{v\}$ and $X_0 \setminus U$. If $v \in X_i$, $0 \leq i < \ell$, then after adding the corresponding edges there is no deleted edge between

$v \in X'_j$ and a vertex of $Y'_{j+1} = U$, whenever $Y_j \neq U$, neither between $v \in X'_{j-1}$ and a vertex of $Y'_j = U$, whenever $Y_j = U$. We need also to show that the rest of the marked vertices still have no deleted edge incident to them as described in Lemma 6.5. Observe that vertices of $X_{i+1}, \ldots, X_\ell$ are marked since they have degree greater than v and, thus, there is no deleted edge between a vertex of X_k and a vertex of Y_{k+1}, for $i+1 \leq k \leq \ell - 1$. This implies that between X'_j and Y'_{j+1} there are no deleted edges and between X'_{j-1} and Y'_j there are no deleted edges since $X'_{j-1} = X_{j-1}$ or $X'_{j-1} = X_{j-1} \cup \{v\}$ and $Y'_j \subseteq Y_j$, for an index j satisfying $i < j \leq \ell$. Moreover if $X_i \neq \{v\}$ then there is no deleted edge incident to a visited vertex of $X'_i \subseteq X_i$ and a vertex of $Y'_{i+1} = Y_{i+1}$. If $X_i = \{v\}$ then $X'_i = X_{i-1}$ and $Y'_{i+1} = Y_{i+1}$ which means that there are some new non-edges in the resulting graph between X'_i and Y'_{i+1} and also between $X'_{i-1} = X_{i-2}$ and $Y'_i = Y_i \cup Y_{i-1}$. However in that case we know that every vertex of $X_{i-1} \cup \ldots \cup X_1 \cup X_0$ has degree strictly less than $v \in X_i$ and, hence, no vertex of that set is marked. Thus at each step of the algorithm there is no deleted edge incident to a marked vertex with one of the two properties given in Lemma 6.5. Therefore by marking all vertices of X we know that at the end there are no deleted edges that can be added by Lemma 6.5 which implies that the computed graph is a minimal chain deletion of G.

For the running time observe that the chain-partition of H and the degree sequence of H are computed in $O(n+m)$ time. We sort the vertices by their degree in H according to a non-increasing order which is required in order to visit the vertices according to their degrees in H. Note that vertices of X that are not visited maintain the same degree since no edge is added incident to them. If $v \in X_\ell$ then we find the proper set $X_0 \setminus U$ by checking its adjacency in G and update at most $d_G(v)$ vertices in $O(d_G(v))$ time. If $v \in X_i$, $0 \leq i < \ell$, then the set $U \subseteq Y_j$, $j > i$ can be found in $O(d_G(v))$ time by scanning $N_G(v)$ constant times. Once we compute the number of adjacent to v vertices in each set Y_k, $k > i$, and then we point out the set $Y_{j'}$ of minimum index j' such that v has at least one non-neighbor in $Y_{j'}$. Since j' is the minimum index we know that in the sets $Y_{i-1}, \ldots, Y_{j'-1}$ there are only adjacent to v vertices. If there are some vertices in $Y_{j'}$ adjacent to v in G then those vertices can be pointed out by scanning $N_G(v)$. Thus if $U = Y_{j'}$ then the added edges are between v and $Y_{i-1}, \ldots, Y_{j'-1}$, whereas if $U \subset Y_{j'}$ then the added edges are between v and $Y_{i-1}, \ldots, Y_{j'} \setminus U$. Adding the deleted edges implies the proper update of the chain partition as described earlier. The number of vertices that need to be updated does not exceed $d_G(v)$ and, thus, each visited vertex v takes $O(d_G(v))$ time. Therefore all steps can be done in $O(n+m)$ time. □

Corollary 6.7. *Any minimal chain deletion of an arbitrary graph can be computed in $O(n+m)$ time.*

Proof. Let $G = (V, E)$ be an arbitrary graph, let $D = E$, and $H = (V, E \setminus D)$. Since an edgeless graph is a chain graph, H is a chain deletion of G. Now, use the algorithm in the proof of Theorem 6.6 to find a minimal chain deletion of G in $O(n+m)$ time. Since $D = E$, we have the possibility of reaching any minimal chain deletion of G. □

6.2 Computing a minimal chain completion of a bipartite graph directly

Here we give an algorithm for computing a minimal chain completion of a bipartite graph G in time $O(n+m)$.

Theorem 6.8. *A minimal chain completion of a bipartite graph can be computed in $O(n+m)$ time.*

Proof. Let G be a bipartite graph and let (X, Y) be a bipartition of G. We start by computing an order of the vertices of X such that $d_G(x_1) \geq d_G(x_2) \geq \ldots \geq d_G(x_{|X|})$ which can be done in $O(n+m)$ time. Initially, we set $H = G$. Starting from x_1, and continuing towards $x_{|X|}$, we add edges to H to make x_i adjacent to all vertices of $N_G(\{x_{i+1}, x_{i+2}, \ldots, x_{|X|}\})$. At the end H remains bipartite and $x_1, \ldots, x_{|X|}$ is a nested neighborhood ordering. Therefore H is a chain graph, and thus a chain completion of G.

Regarding minimality, consider any fill edge xy. We know that $x \in X$ and $y \in Y$. Then y must have a neighbor $z \in X$ such that $yz \in E(G)$, which made it necessary to add the fill edge xy; hence $d_G(x) \geq d_G(z)$. Therefore there is another vertex $w \in N_G(x)$ and $w \notin N_G(z)$. (Note that if $N_G(x) = N_G(z)$ then xy is not a fill edge.) But then the removal of the fill edge xy results in a

Graph Class	Sandwich monotone?	Computing MC	Computing MD	Extracting MC	Extracting MD
chordal	Yes [34]	$O(n^{2.37})$ [22]	$O(\Delta m)$ [11]	$O(nm)$ [10]	Unknown
interval	No	$O(nm)$ [36]	Unknown	$O(n^{11})$ [21]	Unknown
proper interval	No	$O(n+m)$ [32]	Unknown	Unknown	Unknown
split	Yes [18]	$O(n+m)$ [18]	$O(n+m)$ [19]	$O(n+m+\|F\|)$ [18]	$O(n+m)$ [19]
planar	Yes	–	$O(n+m)$ [23]	–	$O(n+m)$ [23]
comparability	No	$O(n^3m)$ [20]	Unknown	Unknown	Unknown
threshold	Yes*	$O(n+m)$*	$O(n+m)$*	$O(n+m+\|F\|)$*	$O(n+m)$*
chain	Yes*	$O(n+m)$*	$O(n+m)$*	$O(n+m+\|F\|)$*	$O(n+m)$*
bipartite	Yes	–	$O(n+m)$*	–	$O(n+m)$*
co-bipartite	Yes	$O(n+m)$*	–	$O(n+m+\|F\|)$*	–

Table 1: Summary of known results for minimal completions and deletions. The input is an arbitrary graph $G = (V, E)$ whereas in the extraction columns the graph $H = (V, E \cup F)$ is assumed to be given as a completion and the graph $H = (V, E \setminus D)$ as a deletion. The asterisk denotes that the corresponding result is obtained in this work, and the dash denotes that the combination is not meaningful. For graph classes that are not listed here, no results are known in any of the columns.

non-chain graph since we obtain a $2K_2$ induced by the vertices $\{w, x, y, z\}$. Hence we cannot remove any single fill edge without destroying chain property, and therefore H is a minimal chain completion of G by Theorem 6.2.

To achieve a time bound of $O(n+m)$, for each vertex $x_i \in X$, starting from v_1, we give its neighbors the label i, $1 \leq i \leq |X|$. Thus at the end, each vertex of Y with label k knows that it must be adjacent to every vertex from x_1 to x_k given in the ordering computed previously of X. Outputting an explicit representation of H (for example, given by its adjacency list representation) requires linear time in the size of H rather than G. Instead, we give a representation of H in $O(n)$ space, which means that we do not explicitly add the fill edges. We output both the sequence $x_1, x_2, ..., x_{|X|}$ and for every vertex of Y its label. This defines uniquely the minimal chain completion H. Therefore all steps can be done in total time $O(n + m)$. □

7 Concluding remarks

In Table 1 we summarize our results by presenting them together with previously known results obtained for other graph classes. For each graph class we give whether the class is sandwich monotone, the best known running time of an algorithm for computing a minimal completion (MC) or a minimal deletion (MD) of an arbitrary graph into this class, and the best known running time of an algorithm for extracting minimal completions or deletions into this class from a given completion or deletion.

It has been shown recently that minimum completions of arbitrary graphs into weakly chordal graphs are NP-hard to compute [7]. We would like to know whether weakly chordal graphs are sandwich monotone. Also, we repeat the open question of [2]: are chordal bipartite graphs sandwich monotone? In addition, we would like to know whether minimum completions of arbitrary bipartite graphs into chordal bipartite graphs are NP-hard to compute.

As a final remark, we mention another possible measure of monotonicity of graph properties. Sandwich monotonicity is a relaxation of monotonicity, hence more graph properties are sandwich monotone than those that are monotone. A natural way to relax sandwich monotonicity is the following: A property $\mathcal{P}$ is called *edge monotone* if for every graph G that satisfies $\mathcal{P}$ there is an edge of G that can be removed without destroying the property $\mathcal{P}$. This also implies that between two graphs satisfying $\mathcal{P}$ there is a sequence of graphs on the same vertex set with one edge difference satisfying $\mathcal{P}$, but now that sequence is not necessarily monotonic (increasing or decreasing). It follows that every sandwich monotone property is edge monotone, hence there are even more properties that are edge monotone. However not all graph classes are edge monotone, hence it might be interesting to characterize edge

monotone graph properties.

References

[1] N. Alon and A. Shapira. Every monotone graph property is testable. In *Proceedings of STOC 2005 - 37th Annual Symposium on Theory of Computing*, pages 128–137, 2005.

[2] M. Bakonyi and A. Bono. Several results on chordal bipartite graphs. *Czechoslovak Math. J.*, 46:577–583, 1997.

[3] J. Balogh, B. Bolobás, and D. Weinreich. Measures on monotone properties of graphs. *Disc. Appl. Math.*, 116:17–36, 2002.

[4] J. Blair, P. Heggernes, and J. A. Telle. A practical algorithm for making filled graphs minimal. *Theoretical Computer Science*, 250:125–141, 2001.

[5] H. L. Bodlaender and A. M. C. A. Koster. Safe separators for treewidth. *Discrete Math.*, 306:337–350, 2006.

[6] V. Bouchitté and I. Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31:212–232, 2001.

[7] P. Burzyn, F. Bonomo, and G. Durán. NP-completeness results for edge modification problems. *Disc. Appl. Math.*, 154:1824–1844, 2006.

[8] K.W. Chong, S.D. Nikolopoulos, and L. Palios. An optimal parallel co-connectivity algorithm. *Theory of Computing Systems*, 37:527 – 546, 2004.

[9] V. Chvátal and P.L. Hammer. Set-packing and threshold graphs. Univ. Waterloo Res. Report, CORR 73–21, 1973.

[10] E. Dahlhaus. Minimal elimination ordering inside a given chordal graph. In *Graph Theoretical Concepts in Computer Science - WG '97*, pages 132–143. Springer Verlag, 1997. LNCS 1335.

[11] P. M. Dearing, D. R. Shier, and D. D. Warner. Maximal chordal subgraphs. *Disc. Appl. Math.*, 20:181-190, 1988.

[12] H. Djidjev. A linear algorithm for finding a maximal planar subgraph. *SIAM J. Disc. Math.*, 20:444 –462, 2006.

[13] F. V. Fomin, D. Kratsch, and I. Todinca. Exact (exponential) algorithms for treewidth and minimum fill-in. In *Proceedings ICALP 2004*, pages 568–580, 2004. Springer LNCS 3142.

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

[15] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[16] P. W. Goldberg, M. C. Golumbic, H. Kaplan, and R. Shamir. Four strikes against physical mapping of DNA. *J. Comput. Bio.*, 2(1):139–152, 1995.

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

[18] P. Heggernes and F. Mancini. Minimal split completions of graphs. In *LATIN 2006: Theoretical Informatics*, pages 592–604. Springer Verlag, 2006. LNCS 3887.

[19] P. Heggernes and F. Mancini. A completely dynamic algorithm for split graphs. Reports in Informatics 334, University of Bergen, Norway, 2006.

[20] P. Heggernes, F. Mancini, and C. Papadopoulos. Making arbitrary graphs transitively orientable: Minimal comparability completions. In *Proceedings of ISAAC 2006 - 17th International International Symposium on Algorithms and Computation*, pages 419–428. Springer Verlag, 2006. LNCS 4288.

[21] P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger. Characterizing minimal interval completions: Towards better understanding of profile and pathwidth. In *Proceedings of STACS 2007 - 24th International Symposium on Theoretical Aspects of Computer Science*, 2007. To appear.

[22] P. Heggernes, J. A. Telle, and Y. Villanger. Computing minimal triangulations in time $O(n^\alpha \log n) = o(n^{2.376})$. *SIAM J. Disc. Math.*, 19:900–913, 2005.

[23] W.-L. Hsu. A linear time algorithm for finding a maximal planar subgraph based on PC-trees. In *Proceedings of COCOON 2005 - 11th International Computing and Combinatorics Conference*, pages 787–797. Springer Verlag, 2005. LNCS 3595.

[24] L. Ibarra. Fully dynamic algorithms for chordal graphs. In *Proceedings of SODA 1999 - 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 923–924, 1999.

[25] H. Ito and M. Yokoyama. Linear time algorithms for graph search and connectivity determination on complement graphs. *Inform. Process. Lett.*, 66:209–213, 1998.

[26] T. Kashiwabara and T. Fujisawa. An NP-complete problem on interval graphs. *IEEE Symp. of Circuits and Systems*, pages 82–83, 1979.

[27] D. König. *Theorie der endlichen und unendlichen Graphen*, Akademische Verlagsgesellschaft, Leipzig, 1936.

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

[29] A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. *Disc. Appl. Math.*, 113:109–128, 2001.

[30] A. Parra and P. Scheffler. Characterizations and algorithmic applications of chordal graph embeddings. *Disc. Appl. Math.*, 79:171–188, 1997.

[31] S.-L. Peng and C.-K. Chen. On the interval completion of chordal graphs. *Disc. Appl. Math.*, 154:1003–1010, 2006.

[32] I. Rapaport, K. Suchan, and I. Todinca. Minimal proper interval completions. In *Proceedings of WG 2006 - 32nd International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 217–228. Springer Verlag, 2006. LNCS 4271.

[33] D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, New York, 1972.

[34] D. Rose, R.E. Tarjan, and G. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:266–283, 1976.

[35] R. Shamir and R. Sharan. A fully dynamic algorithm for modular decomposition and recognition of cographs. *Disc. Appl. Math.*, 136:329 – 340, 2004.

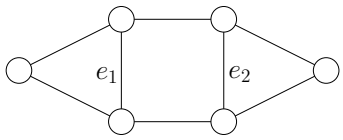
[36] K. Suchan and I. Todinca. Minimal interval completion through graph exploration. In *Proceedings of ISAAC 2006 - 17th International International Symposium on Algorithms and Computation*, pages 517–526. Springer Verlag, 2006. LNCS 4288.

[37] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.

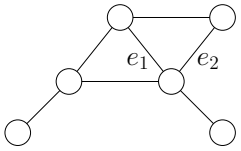
[38] M. Yannakakis. Node deletion problems on bipartite graphs. *SIAM J. Comput.*, 10:310–327, 1981.

Appendix A

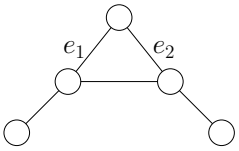
Here we show that the classes of perfect graphs, comparability graphs, interval graphs, proper interval graphs, cographs, trivially perfect graphs, permutation graphs, and $(C_5, C_6, \ldots)$-free graphs are not sandwich monotone. Proper definitions and details about the graph classes can be found in [17]. In all cases we present a graph that belongs to $\mathcal{C}$ with two labeled edges. Removing one of the labeled edges results a graph not in $\mathcal{C}$. But removing both edges results a graph belonging to $\mathcal{C}$. Thus between two graphs of $\mathcal{C}$ on the same vertex set there is no graph of $\mathcal{C}$ which implies that $\mathcal{C}$ is not sandwich monotone.



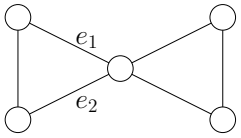
(a) Perfect graphs and comparability graphs



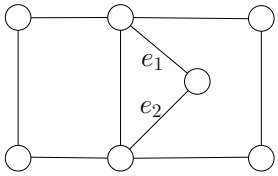
(b) Interval graphs



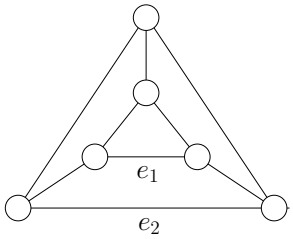
(c) Proper Interval graphs



(d) Cographs and trivially perfect graphs



(e) Permutation graphs



(f) $(C_5, C_6, \ldots)$-free graphs

Figure 1: Examples of graphs showing that their graph classes are not sandwich monotone.