REPORTS IN INFORMATICS

ISSN 0333-3590

Computing and extracting minimal cograph completions in linear time

Daniel Lokshtanov Federico Mancini
Charis Papadopoulos

REPORT NO 352 April 2007

Department of Informatics
UNIVERSITY OF BERGEN
Bergen, Norway

This report has URL http://www.ii.uib.no/publikasjoner/texrap/ps/2007-352.ps

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

Computing and extracting minimal cograph completions in linear time *

Daniel Lokshtanov† Federico Mancini† Charis Papadopoulos†

Abstract

A cograph completion of an arbitrary graph G is a cograph supergraph of G on the same vertex set. Such a completion is called minimal if the set of edges added to G is inclusion minimal. In this paper we characterize minimal cograph completions, and we give the following linear-time algorithms: one for extracting a minimal cograph completion from any given cograph completion of G, and one for directly computing a minimal cograph completion of G with a vertex incremental approach.

1 Introduction

Any graph can be embedded into a cograph by adding edges to the original graph and the resulting graph is called a *cograph completion*, whereas the added edges are called *fill edges*. A cograph completion with the minimum number of edges is called *minimum*, and a cograph completion is called *minimal* if no proper subset of the fill edges result in a cograph when added to the original graph. Computing a minimum completion of an arbitrary graph into a specific graph class is an important and well studied problem with applications in molecular biology, numerical algebra, and more generally areas involving graph modelling with missing edges due to lacking data [10, 17, 20]. Unfortunately minimum completions into most interesting graph classes, *including cographs*, are NP-hard to compute [5, 8, 16, 17, 22]. However, the set of minimum completions is a subset of the set of minimal completions, and hence one can search for a minimum among the set of the minimal ones. In this paper we study minimal cograph completions.

The class of cographs is a well studied graph class that has been discovered in various fields independently, and a large number of papers have been published on it [3]. Many NP-hard problems have polynomial time solutions on cographs. Hence, computing a cograph completion with few edges and solving a hard problem on the computed supergraph is a way of obtaining a good approximation. In a recent WG paper, Nikolopoulos and Palios showed that if the input graph is a graph obtained by adding one edge to a cograph, then a minimum cograph completion of it can be computed in linear time [18].

Given as an input an arbitrary graph, there are two problems related to minimal cograph completions: 1. Computing a minimal cograph completion of the input graph, and 2. Extracting a minimal cograph completion from an arbitrary cograph completion of the input graph. A solution for problem 2 gives a straightforward solution for problem 1 by starting from a complete graph as an initial cograph completion. On the other hand, a solution for problem 1 does not necessarily solve problem 2.Given the motivation that one wants to be able to generate *any* minimal cograph completion in the search of minimum cograph completions, problem 2 is definitely more important and of greater interest. A solution of problem 2 requires a characterization of minimal cograph completions. As an example of usefulness of this, various characterizations of minimal chordal completions have made it possible to design polynomial time approximation algorithms [17] and fast exact exponential time algorithms [9]

*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: {dlo011, federico, charis}@ii.uib.no

for computing minimum chordal completions. A solution of problem 2 also allows the computation of minimal completions that are not far from minimum [2].

In this paper, we solve both problems mentioned above, in linear time. Extracting a minimal completion from a given completion in polynomial time is known only for completions into chordal [1], split [12], and interval [15] graphs. A chordal (split) completion is minimal if and only if no single fill edge can be removed by keeping the completion chordal (split) [21, 12]. This nice property does not hold for cographs. Among graph classes that do not admit this property, a polynomial time algorithm for extracting a minimal completion from a given completion is known only for interval graphs [15]. Hence the algorithm that we present in this paper for the solution of problem 2, is the first linear-time algorithm for extracting a minimal completion into a graph class that does not satisfy the mentioned property.

In this paper, by proving a close relationship between minimal completions and the graph sandwich problem introduced in [11], we reach the conclusion that problem 2 can be solved in polynomial time. However, solving it in linear time requires a series of structural results on minimal cograph completions. First we characterize minimal cograph completions, and then by exploiting this characterization and the structural properties of the unique tree representation (known as the *cotree*) of cographs, we give a linear-time algorithm for extracting a minimal cograph completion from any given one. Furthermore we give a vertex incremental algorithm to compute a minimal cograph completion directly from the input graph in time linear in the size of the computed graph (problem 1).

2 Preliminaries

We consider undirected finite graphs with no loops or multiple edges. For a graph G, we denote its vertex and edge set by $V(G) = V$ and $E(G) = E$, respectively. 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 $G' \subset G$.

The *neighborhood* $N_G(x)$ of a vertex x of the graph G is the set of all the vertices of G which are adjacent to 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, $d_G(x) = |N(x)|$. If $S \subseteq V$, then the neighbors of S, denoted by $N_G(S)$, are given by $\bigcup_{x \in S} N_G(x) \setminus S$. For a pair of vertices x, y of a graph G we call xy a *non-edge* of G if $xy \notin E(G)$. The *complement* $\overline{G}$ of a graph G consists of all vertices and all non-edges of G.

A vertex x of G is *universal* if $N_G(x) = V \setminus \{x\}$ 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. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $V_1 \cap V_2 = \emptyset$, their *union* is $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. Their *join* $G_1 + G_2$ is the graph obtained from $G_1 \cup G_2$ by adding all the edges between the vertices of V_1 and V_2.

A graph G is *connected* if there is a path between any pair of vertices. A *connected component* of a disconnected graph G is a connected subgraph of G with a maximal set of vertices and edges. The *co-connected components* of G are the connected components of $\overline{G}$. By $\mathcal{C}(G)$ and $\widehat{\mathcal{C}}(G)$ we denote the family of the vertex sets of the connected components and co-connected components, respectively, of G. More formally, $\mathcal{C}(G) = \{C_i \mid G[C_i] \text{ is a connected component of } G\}$ and $\widehat{\mathcal{C}}(G) = \{\widehat{C}_i \mid G[\widehat{C}_i] \text{ is a co-connected component of } G\}$.

Given an arbitrary graph $G = (V, E)$ and a graph class Π, a Π *completion* of G is a graph $H = (V, E \cup F)$ such that $H \in \Pi$, and H is a *minimal* Π completion of G if $(V, E \cup F')$ fails to be in Π for every $F' \subset F$. The edges added to the original graph in order to obtain a Π completion are called *fill edges*. Speaking about Π completions of input graphs is only meaningful if every allowed input graph can be embedded in a graph of Π by adding edges. For example, if complete graphs belong to Π then any graph has a Π completion.

2.1 Cographs

The class of cographs, also know as *complement reducible* graphs, is defined recursively as follows:

- A single vertex is a cograph.
- If G_1 and G_2 are cographs, then $G_1 \cup G_2$ is also a cograph.
- If G_1 and G_2 are cographs, then $G_1 + G_2$ is also a cograph.

In this paper we shall also use the following characterization of cographs:

Theorem 2.1 ([6]). *G is a cograph if and only if the complement of any nontrivial connected induced subgraph of G is disconnected.*

Along with other properties, it is known that cographs admit a unique tree representation, called *cotree* [6]. For a cograph G its cotree, denoted by $T(G)$, is a rooted tree having $O(|V|)$ nodes, and $Co(T(G)) = G$. The vertices of G are precisely the leaves of $T(G)$ and every internal node of $T(G)$ is labelled by either 0 (0-node) or 1 (1-node). Two vertices are adjacent in G if and only if their least common ancestor in $T(G)$ is a 1-node. Moreover, if G has at least two vertices then each internal node of the tree has at least two children and any path from the root to any node of the tree consists of alternating 0- and 1-nodes. We assume that a single vertex is represented by a cotree consisting only by itself. Cographs can be recognized and their cotrees can be computed in linear time [7].

For a node t of $T(G)$ we denote by T_t the subtree rooted at t. The set of t's children in $T(G)$ is denoted by $Q(t)$ and the set of leaves of T_t is denoted by $M(t)$. If $S \subseteq V(T(G))$ then $M(S) = \bigcup_{t \in S} M(t)$. Let $Q(t) = \{t_1, \ldots, t_q\}$. If t is a 0-node then $G[M(t)]$ is disconnected with q connected components and $M(t_i) = C_i$, for $C_i \in \mathcal{C}(G[M(t)])$. Otherwise, if t is a 1-node then $G[M(t)]$ is connected with q co-connected components and $M(t_i) = \widehat{C}_i$, for $\widehat{C}_i \in \widehat{\mathcal{C}}(G[M(t)])$.

Let G_1 and G_2 be two vertex-disjoint cographs with cotrees T_1 and T_2, respectively. We define $T_1 + T_2 = T(G_1 + G_2)$ and $T_1 \cup T_2 = T(G_1 \cup G_2)$. Moreover if G is a cograph with a cotree T and u is a child of $root(T)$, then $T - T_u = T(G - M(u))$.

Observation 2.2. *Given two cographs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with cotrees T_1 and T_2 respectively, $T_1 + T_2$ and $T_1 \cup T_2$ can be constructed in $O(|V_1|)$ time.*

Proof. We will prove the statement for $T_1 + T_2$; the same argument holds for $T_1 \cup T_2$. If both $root(T_1)$ and $root(T_2)$ are 0-nodes then we add a 1-node as the root of $T_1 + T_2$ having children $root(T_1)$ and $root(T_2)$. If $root(T_1)$ is 0-node and $root(T_2)$ is a 1-node then in $T_1 + T_2$ we put $root(T_1)$ as a child of $root(T_2)$. If both $root(T_1)$ and $root(T_2)$ are 1-nodes then $T_1 + T_2$ is rooted at $root(T_2)$ and the children of $root(T_1)$ become children of $root(T_2)$. Since in the last case we need to move at most $|V_1|$ pointers, the corresponding construction takes $O(|V_1|)$ time. □

Observation 2.3. *Given a cograph $G = (V, E)$ with cotree $T(G)$ rooted at r and $u \in Q(r)$, $T - T_u$ can be constructed in constant time.*

Proof. If $|Q(r)| > 2$ then we need to remove the pointer from r towards u in $T - T_u$. Otherwise let u' be the sibling of u in T. Then $T - T_u = T_{u'}$ which means that the root of $T - T_u$ is u'. □

2.2 Graph sandwich and minimal completion problems

Here we give a connection between the graph sandwich problem of a graph class and to that of extracting a minimal completion from a completion into a given graph class of an arbitrary graph. The graph sandwich problem for a graph class Π is defined as follows: Given two arbitrary graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ with $E_1 \subseteq E_2$, does there exist a graph $G_3 = (V, E_3)$ belonging to Π such that $E_1 \subseteq E_3 \subseteq E_2$? We refer to G_3 as the *sandwiched* graph. We will show that having a polynomial time algorithm for computing G_30 implies a polynomial time algorithm for computing a minimal Π completion H of G_1 such that $G_1 \subseteq H \subseteq G_2$. Note that G_2 does not necessary belong to the graph class Π and thus the minimal completion problem that we relate to the graph sandwich is more general than the minimal extraction problem from any Π completion.

Lemma 2.4. *Let G_1 and G_2 be two graphs such that $G_1 \subseteq G_2$. If there is a polynomial time algorithm for finding a sandwiched graph G_3 belonging to a graph class Π, then a minimal Π completion H of G_1 such that $G_1 \subseteq H \subseteq G_2$, can be computed in polynomial time.*

Proof. First run the graph sandwich algorithm on G_1 and G_2. If the algorithm does not return any graph G_3, then no Π completion H of G_1 such that $G_1 \subseteq H \subseteq G_2$ exists. Otherwise the algorithm returns a graph $G_3 = (V, E \cup F)$ in Π. G_3 is a Π completion of G_1, but it might not be minimal, so we do the folowing. For each edge $f \in F$, we run the graph sandwich algorithm on G_1 and $G_3' = (V, E \cup F \setminus \{f\})$. If no sandwiched graph is found between G_1 and any possible G_3', then G_3 is minimal because no set $F' \subset F$ exists such that $G_3' = (V, E \cup F')$ is in Π. Therefore G_3 is the minimal Π completion H of G_1. If for at least one fill edge f the algorithm returns a sandwich G_4, then G_3 is not a minimal Π completion of G_1, since G_4 is in Π and $G_4 \subset G_3$. Therefore we iterate the previous procedure on G_1 and G_4 until no sandwiched graph is returned, so that we know that the last graph on which we applied the algorithm is the required minimal Π completion. In total we apply the graph sandwich algorithm, that runs in polynomial time, at most $O(|F|^2)$ times. □

By Lemma 2.4 we conclude that given a polynomial time algorithm for the graph sandwich problem with respect to a graph class Π and substituting G_2 with any Π completion of G_1, we extract a minimal Π completion from G_2 in polynomial time. Therefore all the mentioned problems related to minimal completions for the graph class Π can be solved in polynomial time. Given two arbitrary graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ there is an algorithm for the graph sandwich problem for cographs which runs in $O(|V|(|V|+|E_1|+|E(\overline{G_2})|))$ time [11]. However the running time of the resulting minimal cograph extraction algorithm becomes unpractical, since it requires $O(|V| \cdot |E| \cdot |F|^2)$ time.

3 Characterizing minimal cograph completions

Here we exploit certain properties of cographs in order to characterize minimal cograph completions, and give a linear-time algorithm for deciding whether a given cograph completion is minimal.

Lemma 3.1. *Let $H = (V, E \cup F)$ be a cograph completion of a graph $G = (V, E)$. H is a minimal cograph completion of G if and only if $H[C_i]$ is a minimal cograph completion of $G[C_i]$ for every $C_i \in \mathcal{C}(H)$.*

Proof. Assume $H[C_i]$ is not a minimal cograph completion of $G[C_i]$ for some $C_i \in \mathcal{C}(H)$. Then there exists a graph H' between $G[C_i]$ and $H[C_i]$ that is a cograph and a strict subgraph of $H[C_i]$. Since $H[V \setminus C_i] \cup H'$ is still a cograph and is a strict subgraph of H, H is not minimal. For the other direction, assume that for each $C_i \in \mathcal{C}(H)$, $H[C_i]$ is a minimal cograph completion of $G[C_i]$. Then no subset of the fill edges in each connected component of H can be removed producing a new cograph. Since there are no edges between the connected components of H, it means that no subset of the fill edges can be removed. Hence, H is minimal. □

Let $H = (V, E \cup F)$ be a cograph completion of a graph $G = (V, E)$. If $\overline{H}$ is disconnected, let $\widehat{\mathcal{C}}(H) = \{\widehat{C}_1, \ldots, \widehat{C}_\ell\}$. Given two vertex sets $\widehat{C}_u, \widehat{C}_v \in \widehat{\mathcal{C}}(H)$, we consider the induced subgraph $H[\widehat{C}_u \cup \widehat{C}_v]$. We build a graph G_{uv} by taking $H[\widehat{C}_u \cup \widehat{C}_v]$ and removing all the fill edges between the two vertex sets $\widehat{C}_u$ and $\widehat{C}_v$. More formally, we define

$$G_{uv} = (\widehat{C}_u \cup \widehat{C}_v, E(H[\widehat{C}_u]) \cup E(H[\widehat{C}_v]) \cup E_{uv}),$$

where $E_{uv} = \{xy \mid x \in \widehat{C}_u, y \in \widehat{C}_v, xy \in E\}$. Let us consider now, the subgraphs of H induced by the vertex sets of the connected components of G_{uv}. We define $H_{uv} = \bigcup_{Y_i \in \mathcal{C}(G_{uv})} H[Y_i]$. Notice that if G_{uv} is connected, then $H_{uv} = H[\widehat{C}_u \cup \widehat{C}_v]$; otherwise H_{uv} is disconnected and $G[\widehat{C}_u \cup \widehat{C}_v] \subseteq G_{uv} \subseteq H_{uv} \subset H[\widehat{C}_u \cup \widehat{C}_v]$.

Lemma 3.2. *Let $H = (V, E \cup F)$ be a cograph completion of a graph $G = (V, E)$. H is a minimal cograph completion of G if and only if $H[\widehat{C}_i]$ is a minimal cograph completion of $G[\widehat{C}_i]$, for every $\widehat{C}_i \in \widehat{\mathcal{C}}(H)$ and G_{uv} is a connected graph for any two distinct $\widehat{C}_u, \widehat{C}_v \in \widehat{\mathcal{C}}(H)$.*

Proof. Assume that either $H[\widehat{C}_i]$ is not a minimal cograph completion of $G[\widehat{C}_i]$ for some $\widehat{C}_i \in \widehat{\mathcal{C}}(H)$, or G_{uv} is not a connected graph for some distinct $\widehat{C}_u, \widehat{C}_v \in \widehat{\mathcal{C}}(H)$. We will show that in both cases $H = (V, E \cup F)$ is not a minimal cograph completion of $G = (V, E)$, because we can build a cograph $H' = (V, E \cup F')$, where $F' \subset F$. In the first case there exists a cograph H'_i which is a strict subgraph of $H[\widehat{C}_i]$. Therefore we can define the cograph $H' = H[V \setminus \widehat{C}_i] + H'_i$, that is clearly a strict subgraph of H. For the second case assume that for every $\widehat{C}_i \in \widehat{\mathcal{C}}(H)$, $H[\widehat{C}_i]$ is a minimal cograph completion of $G[\widehat{C}_i]$, but for at least two distinct $\widehat{C}_u, \widehat{C}_v \in \widehat{\mathcal{C}}(H)$, G_{uv} is not connected. Since G_{uv} is not connected, the cograph H_{uv} is a strict subgraph of $H[\widehat{C}_u \cup \widehat{C}_v]$. Thus the graph $H' = H[V \setminus (\widehat{C}_u \cup \widehat{C}_v)] + H_{uv}$ is a cograph by Theorem 2.1 and by construction we know that $G \subseteq H' \subset H$. Hence H is not minimal.

To prove the other direction we show that if $H[\widehat{C}_i]$ is a minimal cograph completion of $G[\widehat{C}_i]$ for each $\widehat{C}_i \in \widehat{\mathcal{C}}(H)$, and G_{uv} is a connected graph for each two distinct $\widehat{C}_u, \widehat{C}_v \in \widehat{\mathcal{C}}(H)$, then H is a minimal cograph completion of G. Since $H[\widehat{C}_i]$ is a minimal cograph completion of $G[\widehat{C}_i]$, no fill edge can be removed from these subgraphs. Assume for the sake of contradiction that H is not minimal. Then there must exist a cograph H' such that $G \subseteq H' \subset H$. By assumption we know that $H'[\widehat{C}_i] = H[\widehat{C}_i]$ for each $\widehat{C}_i \in \widehat{\mathcal{C}}(H)$, and $H'[\widehat{C}_u \cup \widehat{C}_v]$ is connected as a supergraph of G_{uv}, for any two distinct $\widehat{C}_u, \widehat{C}_v \in \widehat{\mathcal{C}}(H)$. We show that H' cannot be a cograph, contradicting the existence of H'. Since $H' \subset H$, there is at least a non-edge in H' between two vertex sets $\widehat{C}_u, \widehat{C}_v \in \widehat{\mathcal{C}}(H)$, so that $H'[\widehat{C}_u]$ is not universal for $H'[\widehat{C}_v]$. This means that $\overline{H'}[\widehat{C}_u \cup \widehat{C}_v]$ is a connected graph because both $\overline{H'}[\widehat{C}_u]$ and $\overline{H'}[\widehat{C}_v]$ are connected graphs (since they are connected components in $\overline{H}$) and there is at least an edges between them. Hence, both $H'[\widehat{C}_u \cup \widehat{C}_v]$ and $\overline{H'}[\widehat{C}_u \cup \widehat{C}_v]$ are connected and H' cannot be a cograph by Theorem 2.1. □

Let $H = (V, E \cup F)$ be a connected cograph completion of G. We know that the root t of $T(H)$ is a 1-node (unless $|V| = 1$). Let t_u, t_v be two nodes of $Q(t)$. If $Q(t_u) \neq \emptyset$ then let $A_u = Q(t_u)$; otherwise let $A_u = \{t_u\}$. Similarly If $Q(t_v) \neq \emptyset$ then let $A_v = Q(t_v)$; otherwise let $A_v = \{t_v\}$. Observe that A_u and A_v contain nodes of $T(H)$. Given the two nodes t_u, t_v we define the graph G^*_{uv} as follows.

$$G^*_{uv} = (A_u \cup A_v, E^*_{uv}),$$

where $E^*_{uv} = \{(a_u, a_v) \in A_u \times A_v \mid M(a_u) \times M(a_v) \cap E \neq \emptyset\}$. In other words, edges are between A_u and A_v, and a vertex a_u of A_u is adjacent to a vertex a_v of A_v if and only if there is an edge $xy \in E$ such that $x \in M(a_u)$ and $y \in M(a_v)$.

Observation 3.3. *Let $T(H)$ be the cotree of a cograph completion H of G and let t be a 1-node of $T(H)$ and $t_u, t_v \in Q(t)$. G^*_{uv} is connected if and only if G_{uv} is connected, where $\widehat{C}_u = M(t_u)$ and $\widehat{C}_v = M(t_v)$. Moreover, for any element $Y_i \in \mathcal{C}(G^*_{uv})$, $M(Y_i) \in \mathcal{C}(G_{uv})$.*

Proof. Let us explain the graph G^*_{uv} with respect to G_{uv}. Given two distinct sets $\widehat{C}_u, \widehat{C}_v \in \widehat{\mathcal{C}}(H[M(t)])$, notice that $G_{uv}[\widehat{C}_u] = H[\widehat{C}_u]$ and $G_{uv}[\widehat{C}_v] = H[\widehat{C}_v]$ are both disconnected cographs or a single vertex, since $H[M(t)]$ is connected. Thus every connected component of $G_{uv}[\widehat{C}_u]$ and $G_{uv}[\widehat{C}_v]$ corresponds to a vertex of A_u and A_v, respectively of G^*_{uv}. Moreover there is at least one edge between two connected components of $G_{uv}[\widehat{C}_u]$ and $G_{uv}[\widehat{C}_v]$ if and only if there is an edge between the corresponding vertices of G^*_{uv}. Hence connectivity of the two graphs is equivalent. □

Theorem 3.4. *Let H be a cograph completion of a graph G and let $T(H)$ be its cotree. H is a minimal cograph completion of G if and only if for every 1-node t of $T(H)$ the graph G^*_{uv} is connected for any two nodes $t_u, t_v \in Q(t)$.*

Proof. Suppose H is a minimal cograph completion of G. Consider a 1-node t of $T(H)$, and let t_u,$t_v \in Q(t)$. $H[M(t)]$ is a minimal cograph completion of $G[M(t)]$, $M(t_u) \in \widehat{\mathcal{C}}(H[M(t)])$ and $M(t_v) \in \widehat{\mathcal{C}}(H[M(t)])$. Now, by Lemma 3.2 and Observation 3.3 G^*_{uv} is connected.

We prove the other direction of the equivalence by induction on $|V(T(H))|$. If $|V(T(H))| = 1$ then G has only one vertex and the result follows. Assume that the statement of the corollary holds whenever $|V(T(H))| < k$. Consider now the case when $|V(T(H))| = k$. By the induction hypothesis $H[M(t_u)]$ is a minimal cograph completion of $G[M(t_u)]$ for every $t_u \in Q(root(T(H)))$. Thus, if $root(T(H)))$ is a 0-node then the result follows by Lemma 3.1. Furthermore if $root(T(H)))$ is a 1-node then we know that G^*_{uv} is connected for every pair of children of the root. Therefore H is a minimal cograph completion of G by Lemma 3.2 and Observation 3.3. □

Remark 3.5. We note that due to Theorem 3.4 a simple linear-time algorithm for deciding whether a given cograph completion H of G is minimal can be obtained: Construct the cotree $T(G)$ of H in linear time [7] and check for every possible pair of siblings (t_u, t_v) in $T(G)$ having 1-node as parent whether the graph G^*_{uv} is connected. A careful construction of G^*_{uv} gives the desired time bound by using the fact that the edges between $M(t_u)$ and $M(t_v)$ are edges of H. However we omit presenting such an algorithm since in the next section (Theorem 4.9) we show a linear-time algorithm for extracting a minimal cograph completion from H which implies that the decision problem can be solved in linear-time as well.

4 Extracting a minimal cograph completion

Here we give a linear-time algorithm to extract a minimal cograph completion from a given cograph completion H of a graph G. Recall that $H_{uv} = \bigcup_{Y_i \in \mathcal{C}(G_{uv})} H[Y_i]$, where Y_i is the vertex set of a connected component of G_{uv}. Our idea relies on Theorem 3.4 when applied on $T(H)$. If for a pair of siblings (u, v) in $T(G)$ the graph G^*_{uv} is connected then we do not remove any fill edge from H. Otherwise, we substitute $H[M(u) \cup M(v)]$ with H_{uv}, which is equivalent to remove the fill edges between the connected components of H_{uv}. Although the idea is quite simple, a linear-time implementation of the algorithm is not straightforward. The algorithm is divided into three parts that we analyze separately: $EMCC$, $Merge$ and *Add*.

$EMCC$ runs itself recursively on the cotree of H and extracts a minimal cograph completion of the connected components of $\overline{H}$ or H, according to whether H is connected or not. If H is disconnected we are done, otherwise $EMCC$ uses $Merge$ to glue together the minimal cograph completions extracted from the co-connected components of H, to form a minimal completion of G. In order to do this, $Merge$ considers one co-connected component at the time, and uses *Add* to remove the unnecessary fill edges between the co-connected component currently considered and the part of H that has been already processed. *Add* is the part of the algorithm that actually implements the general idea we gave before: it substitutes $H[M(u) \cup M(v)]$ with H_{uv} when necessary.

Since the correctness of $Merge$ relies on *Add*, and the correctness of $EMCC$ relies on $Merge$, we present first of all *Add*. Algorithm *Add* extracts a minimal cograph completion from a cograph completion H when the input are a cotree $T(H)$ and a graph G such that: (i) H is connected, and (ii) there is a node $u \in Q(root(T(H))$ such that $H[M(u)]$ and $H - M(u)$ are minimal cograph completions of $G[M(u)]$ and $G - M(u)$, respectively. To prove the correctness of the algorithm we need the following lemma.

Lemma 4.1. *Let $H = (V, E \cup F)$ be a connected cograph completion of a graph $G = (V, E)$, and let $\widehat{C}_u, \widehat{C}_v \in \widehat{\mathcal{C}}(H)$ and $S \subseteq \widehat{\mathcal{C}}(H) \setminus \{\widehat{C}_u, \widehat{C}_v\}$. If $H[S \cup \widehat{C}_u]$ and $H[S \cup \widehat{C}_v]$ are minimal cograph completions of $G[S \cup \widehat{C}_u]$ and $G[S \cup \widehat{C}_v]$, respectively, then $H'[S \cup \widehat{C}_u \cup \widehat{C}_v]$ is a minimal cograph completion of $G[S \cup \widehat{C}_u \cup \widehat{C}_v]$ such that $G[S \cup \widehat{C}_u \cup \widehat{C}_v] \subseteq H'[S \cup \widehat{C}_u \cup \widehat{C}_v] \subseteq H[S \cup \widehat{C}_u \cup \widehat{C}_v]$, where $H' = H_{uv} + (H - (\widehat{C}_u \cup \widehat{C}_v))$.*

Proof. First we prove that H_{uv} is a minimal cograph completion of $G[\widehat{C}_u \cup \widehat{C}_v]$. Let $Y_i \in \mathcal{C}(G_{uv})$. By Lemma 3.1 we need to prove that $H^i_{uv} = H_{uv}[Y_i]$ is a minimal cograph completion of $G[Y_i]$. Each H^i_{uv} is a connected cograph and thus its complement is a disconnected cograph consisting of only two connected components since both $H[\widehat{C}_u]$ and $H[\widehat{C}_v]$ are disconnected and H^i_{uv} is a supergraph of a connected graph. That is, $\widehat{\mathcal{C}}(H^i_{uv}) = \{\widehat{C}_u \cap Y_i, \widehat{C}_v \cap Y_i\}$. Thus we know that $H_{uv}[\widehat{C}_u \cap Y_i] \cup H_{uv}[\widehat{C}_v \cap Y_i]$ is a minimal cograph completion of $G[\widehat{C}_u \cap Y_i] \cup G[\widehat{C}_v \cap Y_i]$. Hence by Lemma 3.2 H^i_{uv} is a minimal cograph completion of $G[Y_i]$, since $G_{uv}[Y_i]$ is a connected component of G_{uv}.

Now observe that $H'[S \cup \widehat{C}_u \cup \widehat{C}_v] = H_{uv} + H[S]$. By construction we know that $H_{uv} + H[S]$ is a cograph. Since no set of the fill edges can be removed from both $H[S \cup \widehat{C}_u]$ and $H[S \cup \widehat{C}_v]$ and by the fact that H_{uv} is a minimal cograph completion of $G[\widehat{C}_u \cup \widehat{C}_v]$, it follows that $H_{uv} + H[S]$ is a minimal cograph completion of $G[S \cup \widehat{C}_u \cup \widehat{C}_v]$. To complete the proof also observe that $G[\widehat{C}_u \cup \widehat{C}_v] \subseteq H_{uv} \subseteq H[\widehat{C}_u \cup \widehat{C}_v]$, by construction. □

Algorithm: Add – `Add` (G, T, u)

Input: A graph $G = (V, E)$, a cotree T rooted at a 1-node, and node u which is a child of the root, such that $Co(T_u)$ and $Co(T - T_u)$ are minimal cograph completions of $G[M(u)]$ and $G[V \setminus M(u)]$, respectively, and $Co(T)$ is a cograph completion of G

Output: A cotree T' such that $Co(T')$ is a minimal cograph completion G and $G \subseteq Co(T') \subseteq Co(T)$

```
1  H = Co(T);
2  P = Q(root(T)) \ {u};
3  foreach v ∈ P do
4      if G*_uv is disconnected then
5          if |Q(root(T))| = 2 then
6              T = T(H_uv);
7          else
8              T = T(H_uv) + ((T − T_u) − T_v);
9          T_u = T(H_uv)  and  u = root(T_u);
10 return T;
```

Lemma 4.2. *Algorithm Add is correct.*

Proof. Let $u_0 = u$, $T_0 = T$, and $H_0 = Co(T)$. Let us define u_i as the node u at the end of the ith iteration of the foreach-loop given between lines 3 and 9. Let $P = \{v_1, \ldots, v_n\}$ such that $v_i \in P$ is the node processed at the ith iteration, $P_i = \{v_1, \ldots, v_i\}$ and define $P_0 = \emptyset$. We partition P_i into two sets of nodes: the set S_i, consisting of the nodes $v_j \in P_i$ such that $G^*_{u_j v_{j+1}}$ is disconnected for $0 \leq j \leq i-1$, and the set $X_i = P_i \setminus S_i$. Let T_i be the cotree T at the end of iteration i. Observe that if $G^*_{u_i v_{i+1}}$ is connected for some $0 \leq i \leq n-1$ then $T_{i+1} = T_i$, $u_{i+1} = u_i$, $S_{i+1} = S_i$, and $X_{i+1} = \{v_{i+1}\} \cup X_i$. If $G^*_{u_i v_{i+1}}$ is disconnected then $u_{i+1} = root(T(H_{u_i v_{i+1}}))$, $S_{i+1} = \{v_{i+1}\} \cup S_i$, and $X_{i+1} = X_i$. In both cases it holds that $T_{i+1} = T(H_{u_i v_{i+1}}) + ((T_i - T_{u_i}) - T_{v_{i+1}})$, whenever $|Q(root(T_{i+1}))| \neq 2$.

First we will prove by induction that for any $0 \leq i < n$, $Q(root(T_i)) = \{u_i\} \cup (P \setminus S_i)$.

Induction hypothesis: $Q(root(T_i)) = \{u_i\} \cup (P \setminus S_i)$ for any $i < n - 1$

Base case: For $i = 0$ we know that $P \cup \{u_0\}$ are children of $root(T_0)$ by the input of the algorithm.

We prove that the induction hypothesis holds for $i + 1$. At iteration $i + 1$ we need to distinguish whether $G^*_{u_i v_{i+1}}$ is connected or not. If true then $T_{i+1} = T_i$ and by the induction hypothesis the result follows. Otherwise, we know by the induction hypothesis that $Q(root(T_i)) = \{u_i\} \cup P \setminus S_i$. Since $i < n - 1$, $|Q(T_i)| > 2$. Hence, $(T_i - T_{u_i}) - T_{v_{i+1}}$ has at least one vertex and by construction we know that T_{i+1} is rooted at a 1-node. Thus $Q(root(T_{i+1})) = \{u_{i+1}\} \cup (Q(root(T_i)) \setminus \{u_i, v_{i+1}\})$, that is $Q(root(T_{i+1})) = \{u_{i+1}\} \cup (P \setminus S_{i+1})$.

Showing the previous argument it follows that $|Q(root(T_i))| = 2$ only if $i = n-1$, since $|P \setminus S_i| > 1$ for any $i < n$. Also, by the definition of $H_{u_i v_{i+1}}$ we know that if $G^*_{u_i v_{i+1}}$ is disconnected then $M(u_{i+1}) = M(u_i) \cup M(S_{i+1})$; otherwise, $M(u_{i+1}) = M(u_i)$ and $S_{i+1} = S_i$. Thus $M(S_i) \subseteq M(u_i)$. Now let $V_i = M(u_i) \cup M(X_i)$ and let $H_i = Co(T_i)$ for any $0 \leq i \leq n$. We will prove that the computed cotree of Algorithm *Add* has the properties as described in the output by induction on $H_i[V_i]$. Note that $V_n = V$ and H_n is the cograph corresponding to the output cotree.

Induction hypothesis: $H_i[V_i]$ is a minimal cograph completion of $G[V_i]$ and $G[V_i] \subseteq H_i[V_i] \subseteq H[V_i]$.

Base case: For $i = 0$ the result follows by the properties given in the input of the algorithm.

Now we prove that the induction hypothesis holds for $i+1$. We know that $H_{i+1} = H_{u_i v_{i+1}} + (H_i - (M(u_i) \cup M(v_{i+1})))$ and either $X_{i+1} = \{v_{i+1}\} \cup X_i$ or $X_{i+1} = X_i$. By the previous argument u_i is a child of $root(T_i)$ and thus $M(u_i), M(v_{i+1}) \in \widehat{\mathcal{C}}(H_i)$. Moreover $H_i[M(v_{i+1}) \cup M(X_i)]$ is a minimal cograph completion of $G[M(v_{i+1}) \cup M(X_i)]$ by the input of the algorithm. Furthermore $H_i[M(u_i) \cup M(X_i)]$ is a minimal cograph completion of $G[M(u_i) \cup M(X_i)]$ by the induction hypothesis. Therefore by applying Lemma 4.1 for $S = M(X_i)$ we obtain the claim properties. □

We are now ready to give Algorithms $EMCC$ and $Merge$.

Algorithm: Extract_Minimal_Cotree_Completion – `EMCC` (G, T)

Input: A graph $G = (V, E)$ and a cotree T such that $Co(T)$ is a cograph completion of G
Output: A cotree T' such that $Co(T')$ is a minimal cograph completion of G, and $G \subseteq Co(T') \subseteq Co(T)$

```
1  P = Q(root(T));
2  foreach c ∈ P do
3      T'_c = EMCC(G[M(c)], T_c);
4      if root(T) is a 1-node then
5          T = T'_c + (T − T_c);
6      else
7          T = T'_c ∪ (T − T_c);
8  T' = T;
9  if root(T) is 1-node then
10     T' = Merge(G, T);
11 return T';
```

Algorithm: Merge – `Merge` (G, T)

Input: A graph $G = (V, E)$, and a cotree T rooted at a 1-node, such that $Co(T)$ is a cograph completion of G and for every $v \in Q(root(T))$, $Co(T)[M(v)]$ is a minimal cograph completion of $G[M(v)]$
Output: A cotree T' such that $Co(T')$ is a minimal cograph completion of G, and $G \subseteq Co(T') \subseteq Co(T)$

```
1  Pick a child u ∈ Q(root(T));
2  if |Q(root(T))| > 2 then
3      T_v = Merge(G − M(u), T − T_u);
4      T = T_u + T_v;
5  T' = Add(G, T, u);
6  return T';
```

Lemma 4.3. *Algorithm Merge is correct.*

Proof. Since T is rooted at a 1-node, it means that $Co(T)$ is connected and it contains at least 2 vertices. Thus $|Q(root(T))| \geq 2$. We will prove the correctness of Algorithm $Merge$ by induction on the size of $Q(root(T))$.

Induction hypothesis: if $|Q(root(T))| \leq k$ then $Co(T')$ is a minimal cograph completion of G with $G \subseteq Co(T') \subseteq Co(T)$.

Base case: Our base case is when $|Q(root(T))| = 2$. Let $Q(root(T)) = \{u, v\}$. The output of $Merge$ will be the same as the output of *Add* on input G, T and u. Since $Co(T_u)$ and $Co(T_v)$ are minimal cograph completions of $G[M(u)]$ and $G[M(v)]$, respectively, by the input of $Merge$, the cotree T' returned by *Add* has the desired properties by Lemma 4.2.

We will now show that the induction hypothesis holds for $|Q(root(T))| = k + 1$. The algorithm will pick a node $u \in Q(root(T))$ and run $Merge$ on $G - M(u)$ and $T - T_u$. The returned cotree T_v represents a minimal cograph completion of $G - M(u)$ such that $G - M(u) \subseteq Co(T_v) \subseteq Co(T - T_u)$, by the induction hypothesis. Observe that $T_u + T_v$ represents a cograph completion of G such that $Co(T_u)$ and $Co(T_v)$ are minimal cograph completions of $G[M(u)]$ and $G - M(u)$, respectively. Moreover u is a child of $T_u + T_v$ since $V(G) \setminus M(u) \neq \emptyset$. Hence by Lemma 4.2 we know that the cotree T' returned by *Add* has the properties described in the output of $Merge$. □

Theorem 4.4. *Algorithm EMCC is correct.*

Proof. We will prove the theorem by induction on the size of V.

Induction hypothesis: T' represents a minimal cograph completion of G for any $|V| \leq k$, such that $G \subseteq Co(T') \subseteq Co(T)$.

Base case: If $|V| = 1$ then T has only one node, hence $Q(root(T)) = \emptyset$ and the algorithm will just output $T' = T$. This is correct because a vertex is a minimal cograph completion of itself.

We will now show that the induction hypothesis still holds for $k+1$. For each $c \in P$ we run Algorithm $EMCC$ on input $(G[M(c)], T_c)$. Since $|M(c)| < k + 1$, the returned cotree T'_c represents a minimal cograph completion of $G[M(c)]$, such that $G[M(c)] \subseteq Co(T'_c) \subseteq Co(T_c)$ by the induction hypothesis. If $root(T)$ is a 1-node then the computed cotree must represent the graph $Co(T'_c) + Co(T - T_c)$; otherwise it must represent the graph $Co(T'_c) \cup Co(T - T_c)$. Notice that the $root(T)$ stays of the same type with the root of the input cotree and T represents a cograph completion of G throughout the whole foreach loop. At the end of the foreach loop, let us call the computed cotree T'. By induction, for each $c \in Q(root(T'))$, $Co(T'_c)$ is a minimal cograph completion of $G[M(c)]$. If $root(T')$ is a 0-node then we output directly T' since by Lemma 3.1 $Co(T')$ is a minimal cograph completion of G, and by construction $G \subseteq Co(T') \subseteq Co(T)$. If $root(T')$ is a 1-node then we return the output of $Merge$ with arguments (G, T'), that has the claimed properties by Lemma 4.3. □

4.1 Running time analysis

Here we analyze the time bounds for extracting a minimal cograph completion by using Algorithm $EMCC$. First of all we will prove the running time needed for Algorithm *Add* which is the most technical part of this section. For clarity reasons we introduce some new notation.

We use the same notation used in the proof of Lemma 4.2. Let $\{v_1, \ldots, v_n\}$ be the order in which Algorithm *Add* processes the nodes in P. Given as an input (G, T, u), let $u_0 = u$ and $T_0 = T$. Let T_{i-1} and $T_{u_{i-1}}$ be, respectively, the cotrees T and T_u at the beginning of the ith iteration of the foreach loop starting at line 3, and let $H_{i-1} = Co(T_{i-1})$, for $1 \leq i \leq n$. Let u_i be the root of T_{u_i}. What we wish to prove first, is that any iteration i of the foreach loop starting at line 3, can be executed in time $O(|M(u_0)| \cdot |M(v_i)|)$. We will prove it for two different cases: $T_{i-1} = T_0$ or $T_{i-1} \neq T_0$.

Lemma 4.5. *At iteration i of the foreach loop starting at line 3 of Algorithm Add, if $T_{i-1} = T_0$ then this iteration can be executed in time $O(|M(u_0)| \cdot |M(v_i)|)$.*

Proof. We know that $T_{u_{i-1}} = T_{u_0}$ which implies $u_{i-1} = u_0$. What we need to show is that we can build and check the connectivity of $G^*_{u_{i-1}v_i} = G^*_{u_0 v_i}$ in time $O(|M(u_0)| \cdot |M(v_i)|)$. We can find the children of u_0 and v_i with the corresponding vertex sets in time $O(|M(u_0)| + |M(v_i)|)$ by traversing the tree.

In order to find all edges between $G[M(u_0)]$ and $G[M(v_i)]$ we need at most time $O(|M(u_0)| \cdot |M(v_i)|)$ by using the adjacency matrix of G. Hence, since $|M(u_0)| + |M(v_i)| \leq |M(u_0)| \cdot |M(v_i)|$, we can build $G^*_{u_0v_i}$ in time $O(|M(u_0)| \cdot |M(v_i)|)$, and store its adjacency list. To check whether it is connected, and contextually find its connected components, we simply run a BFS on $G^*_{u_0v_i}$ in time linear in the size of the graph, namely $O(|M(u_0)| + |M(v_i)| + |M(u_0)| \cdot |M(v_i)|)$. If $G^*_{u_0v_i}$ is connected then there is nothing else to prove. Otherwise we need to modify T_{i-1} in order to obtain $T_i = T(H_{u_{i-1}v_i}) + ((T_{i-1} - T_{u_{i-1}}) - T_{v_i})$, within the same time bound. First we build $T' = (T_{i-1} - T_{u_{i-1}}) - T_{v_i}$ in time $O(|M(u_0)| + |M(v_i)|)$ by Observation 2.3, and then we build the cotree of $H_{u_0v_i}$ which contains $O(|M(u_0)| + |M(v_i)|)$ nodes as follows. Create the 0-node u_i and create a child for each connected component of $G^*_{u_0v_i}$. Such children can be created in time $O(|M(u_0)| + |M(v_i)|)$, since they can be built using the $+$ and $\cup$ operations for cotrees on the subtrees that were rooted at u_0 and v_i. Now create $T_i = T(H_{u_{i-1}v_i}) + T'$ in time $O(M(u_i)) = O(|M(u_0)| + |M(v_i)|)$ by Observation 2.2. Notice that if $|Q(root(T_{i-1}))| = 2$ then we only need to build the cotree of $H_{u_0v_i}$. □

Lemma 4.6. *At iteration i of the foreach loop starting at line 3 of Algorithm Add, if $T_{i-1} \neq T_0$ then this iteration can be executed in time $O(|M(u_0)| \cdot |M(v_i)|)$.*

Proof. We partition the set of children of u_{i-1}, $Q(u_{i-1})$, into two sets of nodes: the set L, such that $M(L) \subseteq M(u_0)$ and the set $R = Q(u_{i-1}) \setminus L$. We also partition $M(R)$ into two sets of vertices W and Z, where $W = M(R) \setminus M(u_0)$ and $Z = M(R) \cap M(u_0)$.

First we prove that $G^*_{u_{i-1}v_i}$ is connected if and only if for every node $l \in L$, there is at least an edge xy in G where $x \in M(l)$ and $y \in M(v_i)$. Notice that $R \neq \emptyset$ since $T_{i-1} \neq T_0$. Now consider any previous iteration j, $1 \leq j < i$, such that $T_{j-1} \neq T_j$ implying that $G^*_{u_{j-1}v_j}$ is disconnected. Thus we have $W = \bigcup_{j<i:T_{j-1}\neq T_j} M(v_j)$. Moreover observe that we do not remove any fill edge incident to two vertices of $M(P)$ and thus $H_i[M(P)] = H[M(P)]$, for any $1 \leq i \leq n$. Hence $G_{u_{i-1}v_i}[W \cup M(v_i)] = \bigcup_{j<i:T_{j-1}\neq T_j} G_{v_jv_i}$. Now, since $H[M(P)]$ is a minimal completion of $G[M(P)]$ by the input of Algorithm *Add*, $G_{v_jv_i}$ is connected for any j by Lemma 3.2 and thus $G_{u_{i-1}v_i}[W \cup M(v_i)]$ is connected as well. We conclude that also $G_{u_{i-1}v_i}[M(R) \cup M(v_i)]$ is connected since every node node $r \in R$ is a 1-node and for any vertex $z \in Z$ there exists a vertex $w \in W$ such that both z and w are in a subtree rooted at some node of R. Hence $G_{u_{i-1}v_i}[M(R) \cup M(v_i)]$ is connected, and so is $G^*_{u_{i-1}v_i}[R \cup A_{v_i}]$ by Observation 3.3, where A_{v_i} is either $\{v_i\}$ or $Q(v_i)$. Therefore since $G^*_{u_{i-1}v_i} = G^*_{u_{i-1}v_i}[L \cup R \cup A_{v_i}]$ the result follows.

Regarding the running time observe that by the previous argument we need to check which edges are in G between the vertices of $M(l)$ and $M(v_i)$ for each $l \in L$. Computing L takes time $O(|M(u_0)|)$, since $|M(L)| = O(|M(u_0)|)$ and every subtree rooted at a node of L has only leaves of $M(u_0)$, implying that a bottom-up traversal of T_{i-1} starting from $M(u_0)$ gives the desired children of u_{i-1}. Thus checking whether $G^*_{u_{i-1}v_i}$ is connected can be done without actually building the graph explicitly, but only checking at most $|M(u_0)| \cdot |M(v_i)|$ pair of vertices in G. Each pair of vertices can be checked in constant time by using the adjacency matrix of G, and thus we require in total time $O(|M(u_0)| \cdot |M(v_i)|)$. Once the connectivity of $G^*_{u_{i-1}v_i}$ is checked, we need to modify the cotree T_{i-1} whenever $G^*_{u_{i-1}v_i}$ is disconnected. Now we show that T_i can be obtained from T_{i-1} in $O(|M(u_0)| + |M(v_i)|)$ time. Let us define L' as the subset of L such that, for every $l' \in L'$ there is no vertex in $M(l')$ that has an edge to some vertex of $M(v_i)$ in G. Then the connected components of $G^*_{u_{i-1}v_i}$ are (i) $G^*_{u_{i-1}v_i}[l']$ for each $l' \in L'$ and (ii) $G^*_{u_{i-1}v_i}[R \cup (L \setminus L') \cup M(v_i)]$. Hence $T_i = T(H_{u_{i-1}v_i}) + ((T_{i-1} - T_{u_{i-1}}) - T_{v_i})$, can be built in the following way. First we build $T' = (T_{i-1} - T_{u_{i-1}}) - T_{v_i}$ in time $O(|M(u_0)| + |M(v_i)|)$ by Observation 2.3. Then we build the cotree of $H_{u_{i-1}v_i}$ by creating a 1-node u_iv_i, and adding $T_{u_{i-1}}$ and T_{v_i} as its subtrees. Create then a 0-node u_i and add an edge from u_iv_i to it which can be done in constant time. Now remove the subtrees rooted at nodes of L' from T_{i-1} and attach them to u_i instead. This operation will take time $O(|M(u_0)|)$ at most. The cotree so built, T_{u_i}, represents correctly the graph $H_{u_{i-1}v_i}$, since L' is a nonempty set by the connected components of $G^*_{u_{i-1}v_i}$. Moreover the children of the root of $T(H_{u_{i-1}v_i})$ are u_iv_i and L'. Therefore building $T_i = T(H_{u_{i-1}v_i}) + T'$ takes $O(|M(u_0)| + |M(v_i)|)$ time since either we make a 1-node having u_i and $root(T')$ as children or we

attach u_i as a child of $root(T')$. Notice that if $|Q(root(T_{i-1}))| = 2$ then we only need to build the cotree of $H_{u_{i-1}v_i}$. □

Lemma 4.7. *On input (G, T, u), the running time of Algorithm Add is $O(|M(u)| \cdot |V \setminus M(u)|)$.*

Proof. Notice that the graph H itself is never used throughout the algorithm, but it is only used to define the graph H_{uv}. Hence we never actually build it. By Lemmas 4.5 and 4.6, we know that each iteration i of the foreach loop starting at line 3 takes time $O(|M(u_0)| \cdot |M(v_i)|)$. Hence, since $P = Q(root(T)) \setminus \{u\}$ and $u_0 = u$, the total running time of Algorithm *Add* is $O(|M(u)| \cdot \sum_{v \in P} |M(v)|)$. □

Lemma 4.8. *On input (G, T) let r be the root of T. Algorithm Merge terminates in $O(\sum_{v \in Q(r)} \sum_{w \in Q(r) \setminus \{v\}} |M(v)| \cdot |M(w)|)$ time*

Proof. We prove by induction on $|Q(r)|$. As $Co(T)$ is a nontrivial connected cograph, $|Q(r)| \geq 2$. Thus the base case is $|Q(r)| = 2$. In this case the main computational part happens in the call $Add(G, T, u)$. Thus when $|Q(r)| = 2$ the statement of the lemma follows directly from Lemma 4.7. Suppose now that the statement of the lemma holds whenever $|Q(r)| < k$ for some $k > 2$. We wish to prove that the statement of the lemma also holds when $|Q(r)| = k$, so let T be a cotree with root r and with $|Q(r)| = k$. By the induction hypothesis the call to $Merge(G - M(u), T - T_u)$ takes at most $O(\sum_{v \in Q(r) \setminus \{u\}} \sum_{w \in Q(r) \setminus \{v,u\}} |M(v)| \cdot |M(w)|)$ time, while the call to $Add(G, T, u)$ takes at most $O(|M(u)| \sum_{v' \in Q(r') \setminus \{u\}} |M(v')|)$ time by Lemma 4.7, where $r' = root(T_u + T_v)$. Thus, since $\sum_{v \in Q(r') \setminus \{u\}} |M(v')| = \sum_{v \in Q(r) \setminus \{u\}} |M(v)|$, the total time is bounded by $O(\sum_{v \in Q(r) \setminus \{u\}} \sum_{w \in Q(r) \setminus \{v,u\}} |M(v)| \cdot |M(w)| + M(u) \sum_{v \in Q(r) \setminus \{u\}} |M(v)|)$, that is $O(\sum_{v \in Q(r)} \sum_{w \in Q(r) \setminus \{v\}} |M(v)| \cdot |M(w)|)$ as claimed. □

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

Proof. We start by computing the cotree of H which can be done in time linear in the size of H [7]. Then we apply Algorithm $EMCC$ on $T(H)$ and by Theorem 4.4 we obtain a cotree $T(H')$ such that H' is minimal cograph completion of G and $G \subseteq H' \subseteq H$. What remains to show is that Algorithm $EMCC$ runs in time linear in the size of H, that is, in $O(|V| + |E| + |F|)$ time. We prove by induction on $|V|$. If $|V| \leq 1$ the statement of the theorem follows trivially. Now, suppose the statement of the theorem holds whenever $|V| < k$ for some $k > 1$ and consider the execution of the algorithm on input (G, T) with $|V| = k$. Let $r = root(T)$. By the inductive hypothesis each recursive call to $EMCC(T_c, G)$ takes $O(|V(H[M(c)])| + |E(H[M(c)])|)$ time. By Observations 2.2 and 2.3, we can implement the assignments on line 5 and 7 in time $O(|M(c)|) = O(|V(H[M(c)])|)$. Thus the first foreach loop takes at most $O(|V| + \sum_{c \in Q(r)} |E(H[M(c)])|)$ time. This means that if r is a 0-node, the result follows. If r is a 1-node, by construction, after the execution of the first foreach loop is done, $Q(r) = Q(root(T'))$ and for every c in $Q(r)$, $M(c)$ in T' is equal to the corresponding set $M(c)$ in T. Following this, Lemma 4.8 guarantees that the call to $Merge(G, T')$ takes at most $O(\sum_{u \in Q(r)} \sum_{v \in Q(r) \setminus \{u\}} |M(u)| \cdot |M(v)|) = O(|E(H) \setminus \bigcup_{c \in Q(r)} E(H[M(c)])|)$ time. Adding up the time for the foreach loop and the call to $Merge$ yields that the total running time is bounded by $O(|V| + |\bigcup_{c \in Q(r)} E(H[M(c)])|) + O(|E(H) \setminus \bigcup_{c \in Q(r)} E(H[M(c)])|) = O(|E| + |V| + |F|)$. □

5 Computing a minimal cograph completion directly

Using Theorem 4.9 we are able to compute a minimal cograph completion of an arbitrary graph G starting from the the complete graph on $|V|$ vertices. However this approach requires $O(|V|^2)$ time. In this section we give a vertex incremental approach to solve the problem of computing a minimal cograph completion without the knowledge of any cograph completion in time linear in the size of the output graph. Cographs are hereditary and and a universal vertex can be added to a cograph without destroying the property of being a cograph. Hence the following is a direct consequence of the results in [13].

Proposition 5.1. *Let H be a minimal cograph completion of an arbitrary graph G, and let G_x be a graph obtained from G by adding a new vertex x adjacent to some vertices of G. There is a minimal cograph completion H_x of G_x such that $H_x - x = H$.*

5.1 Adding a vertex in a cograph

In this section we give an algorithm, namely $MxCC$, that computes a minimal cograph completion based on Proposition 5.1. Hereafter we use $G = (V, E)$ to denote a cograph, unless otherwise specified. Given a vertex x together with a list of vertices $N_x \subseteq V$, we denote by G_x the graph obtained by adding x to G. That is, $G_x = (V \cup \{x\}, E \cup \{xy : y \in N_x\})$. Given a cograph G and a vertex set $N_x \subseteq V$ the algorithm computes a vertex set $S \subseteq V$ such that $N_x \subseteq S$ and $H_x = (V \cup \{x\}, E \cup \{xy : y \in S\})$ is a minimal cograph completion of G_x.

Algorithm: Minimal_x_Cograph_Completion – `MxCC` (G, N_x)
Input: A cograph G, and a set of vertices N_x which are to be made adjacent to a vertex $x \notin V$
Output: An inclusion minimal set $S \subseteq V$ such that $N_x \subseteq S$ and $H_x = (V \cup \{x\}, E \cup \{xy : y \in S\})$ is a cograph
if G is connected **then**
  if there are $\widehat{C}_i \in \widehat{\mathcal{C}}(G)$ and $C_j \in \mathcal{C}(G[\widehat{C}_i])$ s.t. $\widehat{C}_i \cap N_x \neq \emptyset$ and $C_j \cap N_x = \emptyset$ **then**
    $S = MxCC(G[\widehat{C}_i], N_x \cap \widehat{C}_i) \cup (V \setminus \widehat{C}_i)$;
  else
    $S = \bigcup_{\widehat{C}_i \in \widehat{\mathcal{C}}(G) \,:\, \widehat{C}_i \cap N_x \neq \emptyset} \widehat{C}_i$;
else
  if there is a $C_i \in \mathcal{C}(G)$ such that $N_x \subseteq C_i$ **then**
    $S = MxCC(G[C_i], N_x)$;
  else
    $S = \bigcup_{C_i \in \mathcal{C}(G) \,:\, C_i \cap N_x \neq \emptyset} C_i$;
return S;

Observe that the algorithm always terminates because each recursive call takes as an argument a subgraph of G induced by a strict subset of V. We are now ready to prove the correctness of Algorithm $MxCC$.

Lemma 5.2. *Given a cograph G and a set of vertices N_x, Algorithm $MxCC$ returns a set of vertices S such that H_x is a cograph completion of G_x.*

Proof. Observe that as $N_x \subseteq S$ we know that $G_x \subseteq H_x$. Thus it is sufficient to show that H_x is a cograph. We prove this by induction on $|V|$. If $|V| \leq 1$ then H_x has at most two vertices and is a cograph. Assume now that the statement of the lemma holds whenever the input graph has less than k vertices and consider the execution of Algorithm $MxCC(G, N_x)$ on a graph G on k vertices. For each of the following cases we will prove that H_x can be constructed by either taking the union or the join of two cographs. This implies that H_x is a cograph by Theorem 2.1.

First we consider the case when G is connected. If $S = MxCC(G[\widehat{C}_i], N_x \cap \widehat{C}_i) \cup (V \setminus \widehat{C}_i)$ then let S' be the set returned by $MxCC(G[\widehat{C}_i], N_x \cap \widehat{C}_i)$ and let $H'_x = (\widehat{C}_i \cup \{x\}, E(G[\widehat{C}_i]) \cup \{xy : y \in S'\})$. By the induction hypothesis H'_x is a cograph. Thus $H_x = H'_x + G[V \setminus \widehat{C}_i]$ is a cograph. If $S = \bigcup_{\widehat{C}_i \in \widehat{\mathcal{C}}(G) : \widehat{C}_i \cap N_x \neq \emptyset} \widehat{C}_i$ then $H_x = (G[V \setminus S] \cup \{x\}) + G[S]$ is a cograph.

Now consider the case when G is disconnected. If there is a $C_i \in \mathcal{C}(G)$ such that $C_i \subseteq N_x$, then let S' be the set returned by $MxCC(G[C_i], N_x)$ and $H'_x = (C_i \cup \{x\}, E(G[C_i]) \cup \{xy : y \in S'\})$. By the induction hypothesis H'_x is a cograph. Thus $H_x = H'_x \cup G[V \setminus C_i]$ is a cograph. If $S = \bigcup_{C_i \in \mathcal{C}(G) \,:\, C_i \cap N_x \neq \emptyset} C_i$ then $H_x = (G[S] + x) \cup G[V \setminus S]$ is a cograph. □

Observation 5.3. *If G is disconnected, $C_i \in \widehat{\mathcal{C}}(G)$, $N_x \cap C_i = \emptyset$, and $S = MxCC(G, N_x)$ then $S \cap C_i = \emptyset$.*

Proof. If $N_x \subseteq C_j$ for a $C_j \in \widehat{\mathcal{C}}(G)$ and $C_j \neq C_i$ then the claim follows immediately as $S \subseteq C_j$. If $C_i = C_j$ then the call to $MxCC(G[C_j], N_x)$ returns an empty set as $N_x = \emptyset$. Finally, if there is no $C_j \in \widehat{\mathcal{C}}(G)$ so that $N_x \subseteq C_j$, then $S = \bigcup_{C_k \in \mathcal{C}(G)\,:\,C_k \cap N_x \neq \emptyset} C_k$ and the result follows. □

Theorem 5.4. *Given a cograph G and a set of vertices N_x, Algorithm $MxCC$ returns a set of vertices S such that H_x is a minimal cograph completion of G_x.*

Proof. By Lemma 5.2, H_x is a cograph completion of G_x. Thus it is sufficient to show minimality. We prove that H_x is a minimal cograph completion of G_x by induction on $|V|$. If $|V| \leq 1$, H_x is trivially a minimal completion of G_x. Now, assume that the statement of the theorem holds whenever the input graph has less than k vertices and consider the execution of Algorithm $MxCC(G, N_x)$ on a graph G on k vertices. We distinguish between two cases according to the connectivity of G and prove that at each case H_x is a minimal cograph completion of G_x by using Lemmas 3.1 and 3.2.

First we consider the case when G is connected. If there are $\widehat{C}_i \in \widehat{\mathcal{C}}(G)$ and $C_j \in \mathcal{C}(G[\widehat{C}_i])$ such that $\widehat{C}_i \cap N_x \neq \emptyset$ and $C_j \cap N_x = \emptyset$, then let S' be the set returned by $MxCC(G[\widehat{C}_i], N_x \cap \widehat{C}_i)$. Given the set S', consider the graphs $G'_x = (\widehat{C}_i \cup \{x\}, E(G[\widehat{C}_i]) \cup \{xy : y \in \widehat{C}_i \cap N_x\})$ and $H'_x = (\widehat{C}_i \cup \{x\}, E(G[\widehat{C}_i]) \cup \{xy : y \in S'\})$. Now, since $\widehat{C}_i \cap N_x \neq \emptyset$ and $C_j \cap N_x = \emptyset$ we know that $C_j \subset \widehat{C}_i$, $G[\widehat{C}_i]$ is disconnected, and $G[C_j]$ is a connected component of $G[\widehat{C}_i]$. Thus, by Observation 5.3, $C_j \cap S' = \emptyset$. From this it follows that in H_x, x has a non-neighbour in $\widehat{C}_i$. Furthermore, as $G[\widehat{C}_i]$ is a co-connected component of G and x is universal to $V \setminus \widehat{C}_i$ in H_x we conclude that $H[\widehat{C}_i \cup \{x\}]$ is a co-connected component of H_x. Finally, as G is connected and N_x is nonempty, H_x is connected.

We wish to apply Lemma 3.2 in order to show that H_x is a minimal cograph completion of G_x. Let $\widehat{C}_u$ and $\widehat{C}_v$ be the vertex sets of two distinct co-connected components of H_x. If neither $\widehat{C}_u$ nor $\widehat{C}_v$ contains x we know that both $G[\widehat{C}_u]$ and $G[\widehat{C}_v]$ are co-connected components of G. Thus G_{uv} is just $G[C_u \cup C_v]$, so G_{uv} is connected. Now, without loss of generality, $\widehat{C}_u$ contains x. By the discussion in the paragraph above, $\widehat{C}_u = \widehat{C}_i \cup \{x\} = V(H'_x)$. By the induction hypothesis $H_x[\widehat{C}_u]$ is a minimal cograph completion of $G_x[\widehat{C}_u]$. Again $H_x[\widehat{C}_v] = G_x[\widehat{C}_v]$. We now proceed to show the connectivity of G_{uv}. Obviously, $G[\widehat{C}_u \cup \widehat{C}_v \setminus \{x\}] = G[\widehat{C}_i \cup \widehat{C}_v] \subseteq (G_{uv} - x)$. Additionally, as $\widehat{C}_i$ and $\widehat{C}_v$ are vertex sets of co-connected components of G, $G[\widehat{C}_i \cup \widehat{C}_v]$ is connected. As $G[\widehat{C}_i \cup \widehat{C}_v] \subseteq (G_{uv} - x)$, $(G_{uv} - x)$ is connected. But since $\widehat{C}_i \cap N_x \neq \emptyset$, x has a neighbour in C_i so G_{uv} is connected as well. Therefore H_x is a minimal cograph completion of G_x by Lemma 3.2.

Now suppose that there are no $\widehat{C}_i \in \mathcal{C}(\overline{G})$ and $C_j \in \mathcal{C}(G[\widehat{C}_i])$ such that $\widehat{C}_i \cap N_x \neq \emptyset$ and $C_j \cap N_x = \emptyset$. If $N_x = \emptyset$ then $S = \emptyset$ and H_x is trivially a minimal completion of G_x. Otherwise, let us describe the co-connected components of H_x. Let $\widehat{C}_1, ..., \widehat{C}_n$ be the elements of $\widehat{\mathcal{C}}(G)$ such that $\widehat{C}_s \cap N_x \neq \emptyset$ for $1 \leq s \leq n$, and let $\widehat{C}'_{n+1} = \bigcup_{\widehat{C} \in \widehat{\mathcal{C}}(G) : \widehat{C} \cap N_x = \emptyset} \widehat{C}$. In H_x, x is adjacent to every vertex of $\widehat{C}_1, ..., \widehat{C}_n$ and isolated to $\widehat{C}'_{n+1}$. Thus H_x has $n+1$ co-connected components, induced by the sets $\widehat{C}_1, ..., \widehat{C}_n$ and $\{x\} \cup \widehat{C}'_{n+1}$. Observe that $H_x[\{x\} \cup \widehat{C}'_{n+1}] = G_x[\{x\} \cup \widehat{C}'_{n+1}]$ and thus $H_x[\{x\} \cup \widehat{C}'_{n+1}]$ is a minimal cograph completion of $G_x[\{x\} \cup \widehat{C}'_{n+1}]$. The same holds for $H_x[\widehat{C}_s]$, $1 \leq s \leq n$, since x is not contained in $\widehat{C}_s$. Furthermore, for any two distinct integers u and v between 1 and n, the graph G_{uv} is just $G[\widehat{C}_u \cup \widehat{C}_u]$ and is connected. To complete the proof, we prove that the graph obtained from $H_x[\{x\} \cup \widehat{C}'_{n+1}]$, $H_x[\widehat{C}_s]$ and by adding the edges of G_x in between, namely G_{uv}, is connected. The minimality then follows by Lemma 3.2. Notice that every vertex of $\widehat{C}'_{n+1}$ is adjacent to every vertex of $\widehat{C}_s$ as a vertex set of a co-connected component of G. What remains to show is that $G_x[\{x\} \cup \widehat{C}_s]$ is connected for any s between 1 and n. Indeed $G_x[\widehat{C}_s] = G[\widehat{C}_s]$ is disconnected but x is adjacent to at least one vertex of each of $G[\widehat{C}_s]$'s connected components, since by construction $\widehat{C}_s \cap N_x$ is nonempty, and thus by assumption $C_j \cap N_x$ is nonempty for every $C_j \in \mathcal{C}(G[\widehat{C}_s])$. We conclude that H_x is a minimal cograph completion of G_x by Lemma 3.2.

Next we consider the case when G is disconnected. If there is a $C_i \in \mathcal{C}(G)$ such that $N_x \subseteq C_i$ then H_x is a disconnected cograph and $H_x[C_i \cup \{x\}]$ is a minimal cograph completion of $G_x[C_i \cup \{x\}]$ by the induction hypothesis. Moreover for every other $C_j \in \mathcal{C}(G)$ such that $C_j \neq C_i$, $H_x[C_j]$ is a connected component of H_x and $H_x[C_j] = G_x[C_j]$. Thus by Lemma 3.1 H_x is a minimal cograph completion of G_x.

Otherwise, let H'_x be the connected component of H_x containing x, and let $G'_x = G_x[V(H'_x)]$. We prove that H'_x is a minimal cograph completion of G'_x. Then H_x is a minimal cograph completion of G_x by Lemma 3.1. By construction $H_x = G[\bigcup_{C_i \in \mathcal{C}(G)\,:\,C_i \cap N_x \neq \emptyset} C_i] + x$. Also, by assumption $|C_i \in \mathcal{C}(G)\ :\ C_i \cap N_x \neq \emptyset| \geq 2$. Thus, H'_x has two co-connected components, $H'_x[\{x\}]$ and $H'_x - x$. Let $C_u = \{x\}$ and $C_v = V(H'_x) \setminus \{x\}$. $H'_x[C_u] = G'_x[C_u]$ and $H'_x[C_v] = G'_x[C_u]$. We prove that G_{uv} is connected. But x is connected to every connected component of $G'_x - x$ so G'_x is connected. As $G_x \subseteq G_{uv}$, so is G_{uv} and the result follows by Lemma 3.2. □

5.2 Implementing Algorithm $MxCC$ using a cotree representation

In order to obtain a good running time for Algorithm $MxCC$ we give an algorithm that works directly on the cotree of the input graph. That is, we give an algorithm, namely $CMxCC$, that takes as an input the cotree $T(G)$ of a cograph G and a set N_x of vertices in G and returns a set of vertices S of G so that H_x is a minimal cograph completion of G_x. For a node t in $T(G)$, recall that $Q(t)$ is the set of t's children in $T(G)$. Let $Q_x(t) = \{t_i \in Q(t) : M(t_i) \cap N_x \neq \emptyset\}$. Thus $Q_x(t) \subseteq Q(t)$.

Algorithm: Cotree_Minimal_x_Cograph_Completion – `CMxCC` (T, N_x)

Input: A cotree T of a cograph $G = (V, E)$ and a set of vertices N_x which are to be made adjacent to a vertex $x \notin V$

Output: An inclusion minimal set $S \subseteq V$ such that $N_x \subseteq S$ and $H_x = (V \cup \{x\}, E \cup \{xy : y \in S\})$ is a cograph

$r = root(T)$;
if r is a 1-node **then**
  if there is a $t \in Q_x(r)$ such that $\emptyset \subset Q_x(t) \subset Q(t)$ **then**
    $S = CMxCC(T_t, N_x \cap M(t)) \cup (M(r) \setminus M(t))$;
  else
    $S = \bigcup_{t \in Q_x(r)} M(t)$;
else
  if there is a $t \in Q(r)$ such that $Q_x(r) \subseteq \{t\}$ **then**
    $S = CMxCC(T_t, N_x)$;
  else
    $S = \bigcup_{t \in Q_x(r)} M(t)$;
return S;

The correctness of the algorithm follows from the following two observations which imply that Algorithm $CMxCC$ returns the same set S as Algorithm $MxCC$.

Observation 5.5. *Let G be a connected cograph and let r be the root of $T(G)$. There are vertex sets $\widehat{C}_i \in \widehat{\mathcal{C}}(G)$ and $C_j \in \mathcal{C}(G[\widehat{C}_i])$ such that $\widehat{C}_i \cap N_x \neq \emptyset$ and $C_j \cap N_x = \emptyset$ if and only if there is a node $t \in Q_x(r)$ such that $\emptyset \subset Q_x(t) \subset Q(t) \neq \emptyset$.*

Proof. In one direction, suppose r has a child t in $Q_x(r)$ such that $\emptyset \subset Q_x(t) \subset Q(t) \neq \emptyset$. Then $Q_x(t) \setminus Q(t)$ Let $\widehat{C}_i = M(r)$, and let c be an element of $Q(r) \setminus Q_x(r)$. By construction, $G[\widehat{C}_i]$ is a co-connected component of G, $G[C_j]$ is a connected component of $G[\widehat{C}_i]$, $\widehat{C}_i \cap N_x \neq \emptyset$ and $C_j \cap N_x = \emptyset$. In the other direction, suppose there are vertex sets $\widehat{C}_i \in \widehat{\mathcal{C}}(G)$ and $C_j \in \mathcal{C}(G[\widehat{C}_i])$ such that $\widehat{C}_i \cap N_x \neq \emptyset$ and $C_j \cap N_x = \emptyset$. As $\widehat{C}_i \in \widehat{\mathcal{C}}(G)$, r has a child t such that $\widehat{C}_i = M(t)$. Furthermore, as there is a

$C_j \in \mathcal{C}(G[\widehat{C}_i])$ such that $\widehat{C}_j \cap N_x = \emptyset$, t has a child c with $M(c) = C_j$ and thus $c \in Q(t) \setminus Q_x(t)$ and thus $Q_x(t) \subset Q(t)$. As $M(t) \cap N_x \neq \emptyset$ it follows that $t \in Q_x(r)$ and $\emptyset \subset Q_x(t)$. □

Observation 5.6. *Let G be a disconnected cograph and let r be the root of $T(G)$. There is a set $C_i \in \mathcal{C}(G)$ such that $N_x \subseteq C_i$ if and only if there is a node $t \in Q(r)$ such that $Q_x(r) \subseteq \{t\}$.*

Proof. Suppose there is a set $C_i \in \mathcal{C}(G)$ such that $N_x \subseteq C_i$. Then, let t be the child of r so that $M(t) = C_i$. For any other child t' of r, clearly $M(t') = \emptyset$. Thus $Q_x(r) \subseteq \{t\}$. In the other direction, suppose $Q_x(r) \subseteq \{t\}$ for some child t of r. Let $C_i = M(t)$. We know that $C_i \in \mathcal{C}(G)$. We prove that for every $C_j \in \mathcal{C}(G)$ so that $C_j \neq C_i$, $C_j \cap N_x = \emptyset$. Suppose for contradiction that $C_j \cap N_x \neq \emptyset$. Let c be the child of r so that $M(c) = C_j$. Clearly $c \neq t$ and $c \in Q_x(r)$ contradicting that $Q_x(r) \subseteq \{t\}$. □

We are now ready to give a bound on the running time for computing a minimal cograph completion H_x of G_x.

Theorem 5.7. *Given a cograph G and its cotree $T(G)$, there is an algorithm for computing the set of vertices S that are adjacent to x in a minimal cograph completion of G_x which runs in $O(|S|)$ time.*

Proof. We describe such an algorithm. First we compute the set $Q_x(t)$ for every node t in $T(G)$ and then we apply Algorithm $CMxCC$ on $T(G)$. By the previous arguments and Theorem 5.4 the set S returned by Algorithm $CMxCC$ contains the vertices that are adjacent to x in a minimal cograph completion H_x of G_x. Now we analyze the running time.

In addition to the work described below the algorithm does a constant amount of work. This does not pose any problem if $S \neq \emptyset$. However if $S = \emptyset$ then we need to add a constant to the running time bound to cover this case. If $S = \emptyset$ it is easy to see that the algorithm requires constant time. Thus, in the rest of the proof we will assume that $S \neq \emptyset$.

Let us show first that given the set $Q_x(t)$ for every node t in $T(G)$, Algorithm $CMxCC$ makes $O(|S|)$ calls. Let $R = \{r_1, r_2, \ldots, r_\ell\}$ where r_i be the root of the subtree considered at the ith call of Algorithm $CMxCC$. Now we show that $\ell = O(|S|)$. The nodes of R form a path in $T(G)$ starting from the root ($= r_1$), since at most one child of an internal node of $T(G)$ is given as an argument in each call of $CMxCC$. Observe that the labels of the internal nodes (0- or 1-nodes) of a cotree $T(G)$ alternate along any path starting at the root. Thus at least $\lfloor \frac{\ell}{2} \rfloor$ nodes of R are 1-nodes. For the 1-nodes the algorithm adds at least one vertex of G in S. Hence $\ell = O(|S|)$.

Next we prove that for every node t in $T(G)$ the set $Q_x(t)$ can be computed in time $O(|S|)$. Before executing Algorithm $CMxCC$ we start from the leaves of $T(G)$ which correspond to the neighbors of x and then in a bottom up fashion we construct the set P of all internal nodes t in $T(G)$ that satisfy $|Q_x(t)| > 0$. We need to show that $|P| = O(|S|)$. Observe that $R \subseteq P$, since $r_{i+1} \in Q_x(r_i)$ where $r_i, r_{i+1} \in R$. Recall that $\ell = |R|$ and $\ell = O(|S|)$. Thus we need to show that $|P \setminus R| = O(|S|)$. For a node t in $P \setminus R$ observe that $M(t) \subseteq S$ since by the algorithm $\bigcup_{t \in Q_x(r)} M(t) \subseteq S$ where $r \in R$. Therefore $|P \setminus R| = O(|S|)$.

Having computed the set $Q_x(t)$ for each node t in $T(G)$ we show that checking the conditions for each case of the algorithm requires constant time. We avoid scanning the children of the root r by traversing nodes of P once more. Let t be a node in P and let r be t's parent in $T(G)$. Note that if $|Q_x(t)| > 0$ then $t \in Q_x(r)$. If t is a 0-node and $0 < |Q_x(t)| < |Q(t)|$ then r marks its child t; if there is another child of r satisfying the previous condition then r marks exactly one of them. Checking the corresponding condition whenever r is a 1-node takes constant time since either r has a marked child or not. Moreover if r is a 0-node then by checking $|Q_x(r)|$ we require constant time to find the unique node, if any, of $Q_x(r)$. Therefore by traversing the P nodes of $T(G)$ twice we compute the set $Q_x(t)$ so that all conditions of Algorithm $CMxCC$ require constant time. Putting everything together implies that all steps can be done in $O(|S| + 1)$ time. □

Now we are ready to show how to compute a minimal cograph completion of an arbitrary graph in time linear in the size of the cograph completion.

Theorem 5.8. *There is an algorithm for computing a minimal cograph completion $H = (V, E \cup F)$ of an arbitrary graph $G = (V, E)$ in $O(|V| + |E| + |F|)$ time.*

Proof. Let $n = |V|$. Order the vertices of G from v_1 to v_n, let $V_i = \{v_1, v_2, \ldots v_i\}$ and $G_i = G[V_i]$. Let $H_1 = G_1$ and $S_{i+1} = CMxCC(T_i, N_{G_{i+1}}(v_{i+1}))$ where T_i is the cotree of H_i. Construct H_{i+1} from H_i by adding the vertex v_{i+1} and making v_{i+1} adjacent to S_{i+1}. Obviously, T_1 is the cotree of a minimal cograph completion of G_1. If T_i is the cotree of a minimal cograph completion H_i of G_i, then Theorem 5.4 and Proposition 5.1 yield that T_{i+1} is the cotree of a minimal cograph completion H_{i+1} of G_{i+1}. Thus, by induction, T_n is the cotree of a minimal cograph completion $H_n = H$ of $G_n = G$. Finally, we consider the running time for computing H by using the adjacency list of G. Computing S_{i+1} from T_i takes $O(|S_{i+1}| + 1)$ time by Theorem 5.7, where $S_{i+1} = N_{H_{i+1}}(v_{i+1})$. Note also that T_{i+1} can be computed directly from T_i and S_{i+1} in $O(|S_{i+1}|)$ time since updating the cotree requires $O(d)$ time whenever the addition of a vertex of degree d results in a cograph [7]. Therefore the total running time becomes $\sum_{i=2}^{n} O(|S_i| + 1) = O(|V|) + O(\sum_{i=2}^{n} d_H(v_i)) = O(|V| + |E| + |F|)$. □

6 Concluding remarks

We have given a characterization of minimal cograph completions which enabled us to extract them in linear time from any cograph completion of an arbitrary graph. We also gave a linear-time algorithm for computing a minimal cograph completion directly from the arbitrary graph. Thus we extend the family of graphs that admit linear-time algorithms for both computing and extracting minimal completions into the desired graph class.

Since the cotree is a special instance of the modular decomposition tree, we question whether the characterization given in Theorem 3.4 can be generalized for other classes of graphs. Furthermore, in the spirit of minimum cograph completions, a slight and careful modification of our vertex incremental algorithm implies that computing the minimum number of fill edges which are only incident to the newly added vertex can be done in polynomial time. Thus relaxing the condition on the fill edges, it would be interesting to check whether the minimum cograph completion of the graph that results from a cograph by adding a vertex, can be done in polynomial time. Finally we question whether minimal Π completions, subject to they do not coincide with minimum Π completions, can be computed or extracted in polynomial time for all hereditary graph classes Π that can be recognized in polynomial time.

Acknowledgement

The authors would like to express their thanks to Pinar Heggernes for her helpful suggestions which improved the presentation of the paper.

References

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

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

[3] A. Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: A Survey.* SIAM Monographs on Discrete Mathematics and Applications, 1999.

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

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

[6] D.G. Corneil, Y. Perl, and L.K. Stewart. Complement reducible graphs. *Disc. Appl. Math.*, 3:163 – 174, 1981.

[7] D.G. Corneil, Y. Perl, and L.K. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 14:926 – 934, 1985.

[8] E. El-Mallah and C. Colbourn. The complexity of some edge deletion problems. *IEEE Transactions on Circuits and Systems*, 35:354 – 362, 1988.

[9] 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.

[10] 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.

[11] M. C. Golumbic, H. Kaplan, and R. Shamir. Graph sandwich problems. *J. Algorithms*, 19:449 – 473, 1995.

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

[13] 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.

[14] P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger. Minimal interval completions. In *Algorithms - ESA 2005*, pages 403 – 414. Springer Verlag, 2005. LNCS 3669.

[15] 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.

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

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

[18] S.D. Nikolopoulos and L. Palios. Adding an edge in a cograph. In *Graph Theoretic Concepts in Computer Science - WG 2005*, pages 214 – 226. Springer Verlag, 2005. LNCS 3787.

[19] 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.

[20] 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.

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

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