Computing a Clique Tree with the Algorithm Maximal Label Search

Anne Berry, Geneviève Simonet

algorithms

MDPI

Article

Computing a Clique Tree with the Algorithm Maximal Label Search

Anne Berry [1] and Geneviève Simonet [2,*]

[1] LIMOS (Laboratoire d'Informatique, d'Optimisation et de Modélisation des Systèmes) UMR CNRS 6158, Ensemble Scientifique des Cézeaux, F-63178 Aubière CEDEX, France; berry@isima.fr

[2] LIRMM (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier), 161 Rue Ada, F-34095 Montpellier CEDEX 5, France

* Correspondence: genevieve.simonet@umontpellier.fr; Tel.: +33-64-348-2845

Academic Editor: Qianping Gu

Received: 29 October 2016; Accepted: 16 January 2017; Published: 25 January 2017

Abstract: The algorithm MLS (Maximal Label Search) is a graph search algorithm that generalizes the algorithms Maximum Cardinality Search (MCS), Lexicographic Breadth-First Search (LexBFS), Lexicographic Depth-First Search (LexDFS) and Maximal Neighborhood Search (MNS). On a chordal graph, MLS computes a PEO (perfect elimination ordering) of the graph. We show how the algorithm MLS can be modified to compute a PMO (perfect moplex ordering), as well as a clique tree and the minimal separators of a chordal graph. We give a necessary and sufficient condition on the labeling structure of MLS for the beginning of a new clique in the clique tree to be detected by a condition on labels. MLS is also used to compute a clique tree of the complement graph, and new cliques in the complement graph can be detected by a condition on labels for any labeling structure. We provide a linear time algorithm computing a PMO and the corresponding generators of the maximal cliques and minimal separators of the complement graph. On a non-chordal graph, the algorithm MLSM, a graph search algorithm computing an MEO and a minimal triangulation of the graph, is used to compute an atom tree of the clique minimal separator decomposition of any graph.

Keywords: chordal graph; clique tree; perfect elimination ordering; perfect moplex ordering; Maximal Label Search; LexBFS; MCS

1. Introduction

Chordal graphs form an important and well-studied graph class, have many characterizations and properties and are used in many applications. From an algorithmic point of view, connected chordal graphs are endowed with a compact representation as a clique tree, which organizes both the maximal cliques (which are the nodes of the tree) and the minimal separators (which label the edges): in a chordal graph, a minimal separator is the intersection of two maximal cliques, so each minimal separator is a clique (a characterization of chordal graphs [1]). Since a connected chordal graph has at most n maximal cliques, a clique tree has at most n nodes and less than n edges, a very efficient representation of the underlying chordal graph.

A PEO (perfect elimination ordering) of a graph is an ordering of its vertices obtained by successively removing a simplicial vertex of the current graph (a vertex is simplicial if its neighborhood is a clique). A clique tree of a chordal graph can be computed efficiently using the characterization of a chordal graph as a graph, which has a PEO [2]. Gavril [3] showed how to compute a clique tree from an arbitrary PEO of the input chordal graph. Blair and Peyton [4] proposed an algorithm based on the search algorithm MCS (Maximum Cardinality Search) from [5], using the properties of both the PEO and the labels computed by MCS to build a clique tree in a simple and elegant way. The algorithm MCS numbers the vertices using labels that count the number of

processed neighbors. MCS, as well as its famous cousin LexBFS (Lexicographic Breadth-First Search) from [6] were originally tailored to compute a PEO of the input graph if it is chordal, thus leading to an efficient algorithm recognizing chordal graphs.

Recent work by Kumar and Madhaven [7] showed how MCS defines the minimal separators of a chordal graph. LexBFS was also shown to scan both the maximal cliques and the minimal separators of a chordal graph [8].

This family of search algorithms has been recently extended by Corneil and Krueger [9], who introduced LexDFS (Lexicographic Depth-First Search) and MNS (Maximal Neighborhood Search). All of these algorithms function on the same basic principle: they use a labeling process to compute an ordering of the vertices of the input graph. All vertex labels are initialized with the same initial label. At each iteration of the algorithm, a yet unnumbered vertex with a maximal label is chosen, and the labels of its yet unnumbered neighbors are increased.

Berry, Krueger and Simonet [10] introduced the algorithm MLS (Maximal Label Search) as a generalization of these algorithms. MLS takes as input a graph G and a labeling structure $\mathcal{L}$ and computes an ordering of the vertices of G, which is a PEO of G if G is chordal. A condition on the definition of a labeling structure from [10] ensures that MLS computes a PEO of a chordal graph. A still more general labeling search algorithm called GLS (General Label Search) was defined in [11] from a more general definition of a labeling structure, which captures classical categories of graph searches (general search, breadth-first and depth-first searches) in addition to the graph searches derived from MLS.

The question we address in this paper is to determine in which cases MLS can be used to build a clique tree. Our goal is not to improve the time complexity of computing a clique tree, since this complexity is already known to be linear, but to further investigate the algorithm MLS and the properties of the labeling structures involved, in order to determine how MLS can be used to compute a clique tree.

To accomplish this, we first focus on the algorithm from [4], extended-MCS, which computes a clique tree of a chordal graph by computing the maximal cliques of H one after another. The beginning of a new clique is detected by a condition on the labels: as long as the label of the vertex that has just been chosen is strictly greater than the label of the previous vertex, the current clique is increased ; otherwise, a new clique is started. This leads to the two following questions: (1) does MLS compute the maximal cliques one after another? (2) does MLS detect new cliques by a condition on labels?

We first consider Question (1). We know that a clique tree of a chordal graph H can be computed from any PEO of H [3], but not necessarily by processing one clique after another. We will show that the maximal cliques are computed one clique after another if the PEO is a PMO (perfect moplex ordering) of the input chordal graph. A moplex is a clique module whose neighborhood is a minimal separator (thus, in a chordal graph, the closed neighborhood of a moplex is a maximal clique), and a PMO of a graph is an ordering of its vertices obtained by successively removing each vertex of a simplicial moplex of the current graph until it is a clique (a moplex of a chordal graph is necessarily simplicial). Berry and Bordat [8] showed that LexBFS ends on a moplex of the input graph whether it is chordal or not, which implies that LexBFS computes a PMO of the input graph if it is chordal. Berry, Blair, Bordat and Simonet [12] proved the more general result that any instance of the algorithm MLS with totally ordered labels computes a PMO of a chordal graph. Berry and Pogorelcnik [13] showed that MCS and LexBFS can be modified to compute the minimal separators and the maximal cliques of a chordal graph, using the fact that the computed ordering is a PMO of this graph, thus extending the result for MCS from [4]. Xu, Li and Liang [14] showed that LexDFS ends on a moplex of the input graph whether it is chordal or not, thus extending the result for LexBFS from [8]. We show in this paper that a slight variant of MLS computes a PMO of the input chordal graph for each labeling structure. As this variant is equivalent to MLS if the order on the labels is total, this generalizes the result from [12] that MLS used with totally ordered labels computes a PMO of a chordal graph.

Concerning Question (2), we give a necessary and sufficient condition on a labeling structure for MLS to detect new cliques by a condition on labels. Because this condition is not satisfied by the labeling structure associated with the algorithm LexDFS, the LexDFS labels do not detect new cliques, contrary to what is claimed in [14].

We then go on to examine what happens when the graph is not chordal.

When a graph is chordal, the minimal separators are cliques. When the graph fails to be chordal, it still may have clique minimal separators. The related graph decomposition (called clique minimal separator decomposition) has given rise to recent interest; see for example [15–19]. This decomposition results in a set of overlapping subgraphs called atoms, characterized as the maximal connected subgraphs containing no clique separator.

Recent work by Berry, Pogorelcnik and Simonet [20] has shown that the atoms of a connected graph can be organized into a tree similar to a clique tree, called an atom tree: the nodes are the atoms, and the edges represent the clique minimal separators of the graph. As is the case for a clique tree, an atom tree has at most n nodes and less than n edges.

The work in [20] showed how an atom tree can be computed from a clique tree of a minimal triangulation (which is a minimal embedding of a graph into a chordal graph), providing an algorithm based on MCS-M [21], the triangulating counterpart of MCS, to build an atom tree.

In this paper, we further address the question of using MLSM, the triangulating counterpart of MLS defined in [10] which is a generalization of MCS-M, to build this atom tree.

The main contributions of this paper are the following. We provide a characterization of PMOs (Characterization 3), which is helpful in proofs concerning PMOs. We show how a clique tree of a chordal graph H can be computed from a PMO of H by building the maximal cliques one after another (Corollary 1) and how the general algorithm MLS can be modified in order to compute a PMO (the algorithm moplex-MLS), and a PMO, a clique tree and the minimal separators (the algorithm MLS-CliqueTree) of any chordal graph for any labeling structure. We also give a necessary and sufficient condition on a labeling structure (DCL) for new cliques to be detected by a condition on labels (Theorem 4). We further modify MLS to compute a PMO and a clique tree of the complement graph, and we show that new cliques in the complement graph can be detected by a condition on labels for each labeling structure (the algorithm complement-DCL-MLS-CliqueTree). We also provide a linear time algorithm computing a PMO and the generators of the maximal cliques and minimal separators w.r.t. this PMO of the complement graph (the algorithm complement-DCL-MLS-generators). We finally extend the results from MLS to MLSM to compute an atom tree of any graph (the algorithm DCL-MLSM-AtomTree).

The paper is organized as follows. Section 2 gives the preliminaries of the paper. Section 3 explains how MLS can be modified into an algorithm computing a PMO and a clique tree of a chordal graph. In Section 4, MLS is used to compute a clique tree of the complement graph. Section 5 gives some extensions: the use of MLSM to compute an atom tree of a graph and some counterexamples when running MLS on a non-chordal graph. The final section concludes the paper.

2. Preliminaries

All graphs in this work are connected, undirected and finite. A graph is denoted by $G = (V, E)$, with $|V| = n$ and $|E| = m$. $E(G)$ is the set of edges of G. $\overline{G}$ denotes the complement of G. The neighborhood of a vertex x in a graph G is $N_G(x)$, or $N(x)$ if the context is clear; the closed neighborhood of x is $N[x] = N(x) \cup \{x\}$. The neighborhood of a subset X of V is $N(X) = (\cup_{x \in X} N(x)) \setminus X$, and its closed neighborhood is $N[X] = N(X) \cup X$. A clique is a set of pairwise adjacent vertices; we say that we saturate a set X of vertices when we add all of the edges necessary to turn X into a clique. A vertex (or a subset of V) is simplicial if its neighborhood is a clique. A module is a subset X of V, such that $\forall x \in X, N(x) \setminus X = N(X)$. $G(X)$ denotes the subgraph of G induced by the subset X of V, but we will sometimes just denote this by X. The reader is referred to [22,23] for classical graph definitions and results.

2.1. Separators

A set S of vertices of a connected graph G is a separator if $G(V \setminus S)$ is not connected. A separator S is an xy-separator if x and y lie in two different connected components of $G(V \setminus S)$. S is a minimal xy-separator if S is an xy-separator and no proper subset of S is also an xy-separator. A separator S is said to be minimal if there are two vertices x and y such that S is a minimal xy-separator. Equivalently, S is a minimal separator if and only if $G(V \setminus S)$ has at least two connected components C_1 and C_2, such that $N_G(C_1) = N_G(C_2) = S$. A moplex is a clique module X whose neighborhood $N(X)$ is a minimal separator.

2.2. Chordal Graphs and Clique Trees

A graph is chordal (or triangulated) if it contains no chordless-induced cycle of length four or more. A graph is chordal if and only if all of its minimal separators are cliques [1]. It follows that for each moplex X of a chordal graph H, X is simplicial and $N[X]$ is a maximal clique of H.

A chordal graph is often represented by a clique tree:

Definition 1. *Let $H = (V, E)$ be a connected chordal graph. A clique tree of H is a tree $T = (V_T, E_T)$ such that V_T is the set of maximal cliques of H and for any vertex x of H, the set of nodes of T containing x induces a subtree of T.*

Characterization 1. *[4] Let H be a connected chordal graph; let T be a clique tree of H; and let S be a set of vertices of H; then, S is a minimal separator of H if and only if there is an edge K_1K_2 of T such that $S = K_1 \cap K_2$.*

Every chordal graph has at least one clique tree, which can be computed in linear time with the nodes labeled by the maximal cliques and the edges labeled by the minimal separators [4].

2.3. Orderings, PEOs and PMOs

An ordering of G is a one-to-one mapping from $\{1, \ldots, n\}$ to V. An ordering α can be defined by the sequence $(\alpha(1), \ldots, \alpha(n))$. A perfect elimination ordering (PEO) of G is an ordering $\alpha = (x_1, \ldots, x_n)$ of G such that for each $i \in [1, n]$, x_i is a simplicial vertex of $G(\{x_i, \ldots, x_n\})$. G is chordal if and only if it has a PEO. An ordering α of G is compatible with an ordered partition $(X_1, \ldots, X_k)$ of V if for each i in $[1, k-1]$, for each u in X_i and each v in X_{i+1}, $\alpha^{-1}(u) < \alpha^{-1}(v)$. A simple (resp. perfect) moplex partition of G is an ordered partition $(X_1, \ldots, X_k)$ of V such that for each $i \in [1, k-1]$, X_i is a moplex (resp. simplicial moplex) of $G(\cup_{i \leq j \leq n} X_j)$ and X_k is a clique of G. Thus, a simple moplex partition of a chordal graph H is a perfect moplex partition of H. A perfect moplex ordering (PMO) of G is an ordering of G compatible with a perfect moplex partition of G. A PMO of G is a PEO of G, and G is chordal if and only if it has a PMO [24].

2.4. Minimal Triangulations, MEOs and MMOs

A triangulation of a graph $G = (V, E)$ is a chordal graph in the form $H = (V, E + F)$. F is the set of fill edges in H. The triangulation is minimal if for any proper subset F' of F, the graph $(V, E + F')$ fails to be chordal. Given an ordering $\alpha = (x_1, \ldots, x_n)$ of G, the graph G^+_α is defined as follows: initialize the current graph G' with G and the set F_α with the empty set; then, for each i from one to n, let F_i be the set of edges necessary to saturate the neighborhood of x_i in G'; add the edges of F_i to G' and to F_α; and remove x_i from G'. $G^+_\alpha = (V, E + F_\alpha)$ is a triangulation of G, with α as a PEO. α is called a minimal elimination ordering (MEO) of G if there is no ordering β of G such that $F_\beta \subset F_\alpha$. α is an MEO of G if and only if G^+_α is a minimal triangulation of G. A minimal triangulation of G is obtained by replacing vertex x_i by a moplex X_i of G' in the preceding process, which is formally defined as follows. A minimal moplex partition of G is an ordered partition $(X_1, \ldots, X_k)$ of V such that for each $i \in [1, k-1]$, X_i is a moplex of G_i and X_k is a clique of G_k, where the graphs G_i are defined

by induction: $G_1 = G$ and for each $i \in [1, k-1]$; G_{i+1} is obtained from G_i by saturating $N_{G_i}(X_i)$ and removing X_i. Thus, a minimal moplex partition of a chordal graph H is a perfect moplex partition of H. A minimal moplex ordering (MMO) of G is an ordering of G compatible with a minimal moplex partition of G. An MMO of G is an MEO of G [24].

2.5. Clique Minimal Separators and Atom Trees

The atoms of a graph G are the subsets of V obtained by clique minimal separator decomposition. The reader is referred to [15,18,19] for full details on the decomposition by clique separators and by clique minimal separators. An atom of a connected graph G is a subset of V inducing a connected subgraph having no clique separator and being inclusion-maximal for this property. The atoms of a chordal graph are its maximal cliques. The atoms of a graph can be organized into an atom tree in the same way as the maximal cliques of a chordal graph are organized into a clique tree [20].

Definition 2. *[20] Let $G = (V, E)$ be a connected graph. An atom tree of G is a tree $T = (V_T, E_T)$ such that V_T is the set of atoms of G and for any vertex x of G; the set of nodes of T containing x induces a subtree of T.*

Characterization 2. *[20] Let G be a connected graph; let T be an atom tree of G; and let S be a set of vertices of G; then S is a clique minimal separator of G if and only if there is an edge A_1A_2 of T, such that $S = A_1 \cap A_2$.*

2.6. Algorithms MLS and MLSM

The algorithm MLS (Algorithm 1), where MLS stands for (Maximal Label Search, generalizes the well-known algorithms MCS from [5], LexBFS from [6] and LexDFS and MNS from [9]. The algorithm MLS takes a graph H and a labeling structure $\mathcal{L}$ as input and yields an ordering of H as output, which is a PEO of H if H is chordal. It can be seen as a generic algorithm with parameter $\mathcal{L}$ whose instances are the algorithms $\mathcal{L}$-MLS for each labeling structure $\mathcal{L}$, with a graph H as input and an ordering of H as output. In the following definitions, $\mathbb{N}^+$ denotes the set of positive integers.

Definition 3. *A labeling structure is a structure $\mathcal{L} = (L, \preceq, l_0, Inc)$, where:*

- *L is a set (the set of labels),*
- *$\preceq$ is a partial order on L (which may be total or not, with $\prec$ denoting the corresponding strict order),*
- *l_0 is an element of L (the initial label),*
- *Inc (increase) is a mapping from $L \times \mathbb{N}^+$ to L satisfying the following IC (Inclusion Condition): for any subsets I and I' of $\mathbb{N}^+$, if $I \subset I'$, then $lab_{\mathcal{L}}(I) \prec lab_{\mathcal{L}}(I')$, where $lab_{\mathcal{L}}(I) = Inc(\ldots(Inc(l_0, i_1), \ldots), i_k)$, where $I = \{i_1, i_2, \ldots, i_k\}$, with $i_1 > \cdots > i_k$.*

For each X in {MCS, LexBFS, LexDFS, MNS}, the algorithm X is the instance $\mathcal{L}_X$-MLS of the algorithm MLS, where $\mathcal{L}_X$ is the labeling structure $(L, \preceq, l_0, Inc)$ defined as follows.

$\mathcal{L}_{MCS}$: $L = \mathbb{N}^+ \cup \{0\}$, $\preceq$ is $\leq$ (a total order), $l_0 = 0$, $Inc(l, i) = l + 1$.

$\mathcal{L}_{LexBFS}$: L is the set of lists of elements of $\mathbb{N}^+$, $\preceq$ is the usual lexicographic order (a total order), l_0 is the empty list, $Inc(l, i)$ is obtained from l by appending i to the end of the list.

$\mathcal{L}_{LexDFS}$: L is the set of lists of elements of $\mathbb{N}^+$, $\preceq$ is the lexicographic order where the order on $\mathbb{N}^+$ is the reverse order of the usual one (a total order), l_0 is the empty list, $Inc(l, i)$ is obtained from l by prepending i to the beginning of the list.

$\mathcal{L}_{MNS}$: L is the power set of $\mathbb{N}^+$, $\preceq$ is $\subseteq$ (not a total order), $l_0 = \emptyset$, $Inc(l, i) = l \cup \{i\}$.

Algorithm 1: MLS.

input : a connected graph H and a labeling structure $\mathcal{L} = (L, \preceq, l_0, Inc)$
output: an ordering $\alpha = (x_1, \ldots, x_n)$ of H, which is a PEO of H if H is chordal

$V' \leftarrow \emptyset$; initialize all labels as l_0;
for $i = n$ ***down to*** 1 **do**
  Choose a vertex x in $V \setminus V'$ whose label is maximal;
  $x_i \leftarrow x$;
  foreach y *in* $N_H(x) \setminus V'$ **do**
    $label(y) \leftarrow Inc(label(y), i)$;
  end
  $V' \leftarrow V' \cup \{x\}$;
end

The algorithm MLSM (Algorithm 2), where the final letter M stands for MEO, generalizes the algorithms LEX M from [6] and MCS-M from [21]. It takes a graph G and a labeling structure $\mathcal{L}$ as input and yields an MEO of G and the associated minimal triangulation of G as output. It can be seen as a generic algorithm in the same way as the algorithm MLS. The algorithms LEX M and MCS-M are the instances $\mathcal{L}_{LexBFS}$-MLSM and $\mathcal{L}_{MCS}$-MLSM, respectively, of the algorithm MLSM.

Algorithm 2: MLSM.

input : a connected graph G and a labeling structure $\mathcal{L} = (L, \preceq, l_0, Inc)$
output: an MEO $\alpha = (x_1, \ldots, x_n)$ of G and the associated minimal triangulation $H = G^+_\alpha$

$H \leftarrow G$; $V' \leftarrow \emptyset$; initialize all labels as l_0;
for $i = n$ ***down to*** 1 **do**
  Choose a vertex x in $V \setminus V'$ whose label is maximal;
  $Y \leftarrow \emptyset$;
  foreach y *in* $V \setminus (V' \cup \{x\})$ **do**
    if *there is a path* μ *of length* ≥ 1 *in* $G(V \setminus V')$ *between* x *and* y *such that for each internal vertex* z *of* μ $label(z) \prec label(y)$ **then**
      $Y \leftarrow Y + \{y\}$; $E(H) \leftarrow E(H) \cup \{xy\}$;
    end
  end
  foreach y *in* Y **do**
    $label(y) \leftarrow Inc(label(y), i)$;
  end
  $V' \leftarrow V' \cup \{x\}$;
end

Notation 1. *Let G be a graph; let α be an ordering of G; let $i \in [1, n]$; and let $y \in V$.*

- $N_G^{\alpha,i}(y) = \{z \in N_G(y) \mid \alpha^{-1}(z) > i\}$,
- $N_G^{\alpha+}(y) = N_G^{\alpha,\alpha^{-1}(y)}(y) = \{z \in N_G(y) \mid \alpha^{-1}(z) > \alpha^{-1}(y)\}$,
- $N_G^{\alpha+}[y] = N_G^{\alpha+}(y) \cup \{y\} = \{z \in N_G[y] \mid \alpha^{-1}(z) \geq \alpha^{-1}(y)\}$,
- *a generator of a minimal separator S (resp. maximal clique K) of G w.r.t. α is a vertex x of G*

such that $S = N_G^{\alpha+}(x)$ (resp. $K = N_G^{\alpha+}[x]$).

If α is a PEO of a chordal graph H, then each minimal separator of H has at least one generator w.r.t. α [25], and each maximal clique of H clearly has exactly one generator w.r.t. α, which is the vertex x of K with minimum $\alpha^{-1}(x)$.

Lemma 1. *In an execution of MLS, for each i in $[1, n-1]$ and each y, z in V such that $\alpha^{-1}(y) \leq i+1$ and $\alpha^{-1}(z) \leq i+1$, at the beginning of iteration i (choosing vertex x_i) of the* ***for*** *loop,*

(i) *If $N_G^{\alpha,i}(y) \subset N_G^{\alpha,i}(z)$, then $label(y) \prec label(z)$,*
(ii) *If $N_G^{\alpha,i}(y) \subseteq N_G^{\alpha,i}(z)$, then $label(y) \preceq label(z)$.*

Proof. (i) If $N_G^{\alpha,i}(y) \subset N_G^{\alpha,i}(z)$, then $\alpha^{-1}(N_G^{\alpha,i}(y)) \subset \alpha^{-1}(N_G^{\alpha,i}(z))$, and therefore, by IC, $label(y) = lab_{\mathcal{L}}(\alpha^{-1}(N_G^{\alpha,i}(y))) \prec lab_{\mathcal{L}}(\alpha^{-1}(N_G^{\alpha,i}(z))) = label(z)$. The proof of (ii) is similar. □

3. MLS and Clique Trees

In this section, we show how the general algorithm MLS can be used to compute a clique tree and the minimal separators of a chordal graph, thus generalizing the results given in [4] for MCS and in [13] for LexBFS. We will accomplish this by applying successive modifications on an algorithm computing a clique tree from a PEO of a chordal graph.

3.1. Clique Tree from a PEO

The algorithm CliqueTree (Algorithm 3) from Spinrad ([26], p. 258) is a slight variant of an algorithm originally given by Gavril [3] to compute a clique tree of a connected chordal graph from an arbitrary PEO of this graph.

Algorithm 3: CliqueTree.

input : a connected chordal graph H and a PEO $\alpha = (x_1, \ldots, x_n)$ of H
output: a clique tree T and the set Sep of minimal separators of H

$V' \leftarrow \emptyset$; $s \leftarrow 1$; $K_1 \leftarrow \emptyset$; $E \leftarrow \emptyset$; $Sep \leftarrow \emptyset$;
for $i = n$ ***down to*** 1 **do**
  $x \leftarrow x_i$; $S \leftarrow N_H(x) \cap V'$; // $S = N_H^{\alpha+}(x)$
  if $i = n$ **then**
    $p \leftarrow 1$;
  end
  else
    $k \leftarrow min\{j, \alpha(j) \in S\}$; // $S \neq \emptyset$ because H is connected, α is a PEO of H and $i < n$
    $p \leftarrow clique(\alpha(k))$;
  end
  if $K_p = S$ **then**
    $clique(x) \leftarrow p$;
  end
  else
    $s \leftarrow s+1$; $K_s \leftarrow S$; // start a new clique
    $E \leftarrow E \cup \{(p, s)\}$; $Sep \leftarrow Sep \cup \{S\}$; // $S = K_p \cap K_s$
    $clique(x) \leftarrow s$;
  end
  $K_{clique(x)} \leftarrow K_{clique(x)} \cup \{x\}$; // increase clique
  $V' \leftarrow V' \cup \{x\}$;
end
$T \leftarrow (\{K_1, \ldots, , K_s\}, \{K_p K_q, \ (p, q) \in E\})$;

Theorem 1. *[3,26] The algorithm CliqueTree computes a clique tree and the set of minimal separators of a connected chordal graph H from a PEO of H in linear time.*

Note that the algorithms from [3,26] only compute a clique tree of the input graph. By Characterization 1, the algorithm CliqueTree correctly computes the set of minimal separators, and it does so in linear time using a search/insert structure for Sep , which allows checking for the presence of a set S and inserting it in $O(|S|)$ time. The proofs given in [3,26] implicitly use Invariant 1 below, which is explicitly stated and proven here, since it will be used later in this paper. To prove it, we will use the following lemma.

Lemma 2. *Let H be a connected chordal graph; let x be a simplicial vertex of H; let $H' = H(V \setminus \{x\})$; let T' be a clique tree of H'; let K be a node of T' containing $N_H(x)$; let $K' = N_H[x]$; and let T be the tree obtained from T' by replacing node K by K' (with the same neighbors in T as in T') if $K = N_H(x)$ and by adding node K' and edge KK' otherwise. Then, T is a clique tree of H.*

Proof. Let us show that the nodes of T are the maximal cliques of H. As x is simplicial in H, K' is the unique maximal clique of H containing x. Each maximal clique of H different from K' is a maximal clique of H', and each maximal clique of H' different from $N_H(x)$ is a maximal clique of H. It follows that the nodes of T are the maximal cliques of H. It remains to show that for each vertex y of H, the subgraph T_y of T induced by the set of nodes of T containing y is connected. If $y = x$, then T_y is reduced to node K'; otherwise, T_y is either equal to T'_y or obtained from T'_y by replacing node K by node K' or by nodes K and K' and edge KK'. Hence, T_y is connected. □

Invariant 1. *The following proposition is an invariant of the **for** loop of the algorithm CliqueTree:*

(a) $(\{K_1, \ldots, K_s\}, \{K_pK_q,\ (p,q) \in E\})$ *is a clique tree of* $H(V')$,
(b) Sep *is the set of minimal separators of* $H(V')$,
(c) $\forall y \in V'$, $clique(y) \in [1,s]$ *and* $N_H^{\alpha+}[y] \subseteq K_{clique(y)}$.

Proof. The proposition clearly holds at the initialization step of the **for** loop. Let us show that it is preserved by each iteration of this loop. It is clearly preserved by iteration n, i.e., the iteration where $i = n$. Let us show that it is preserved by iteration i, with $1 \leq i < n$. Let (a1) (resp. (b1), (c1)) denote item (a) (resp. (b), (c)) at the beginning of iteration i (which is supposed to be true), and let x, V', S, s, k and p be the values of these variable at the end of iteration i. As α is a PEO of H, x is a simplicial vertex of $H(V')$, so S is a clique of H. It follows by definition of k that $S \subseteq N_H^{\alpha+}[\alpha(k)]$; hence, by (c1) $S \subseteq K_p$ with $p \in [1,s]$. Thus, by (a1), K_p is a maximal clique of $H(V' \setminus \{x\})$ containing S. It follows from Lemma 2 and Characterization 1 that (a) and (b) are preserved. It remains to show that (c) is preserved. It is the case since for each $y \in V' \setminus \{x\}$, $clique(y)$ and $N_H^{\alpha+}[y]$ are unchanged, whereas s and $K_{clique(y)}$ can only become bigger at iteration i, and for $y = x$, $clique(x)$ is either equal to p or to s with $p \in [1,s]$; and $N_H^{\alpha+}[x] = S \cup \{x\} = K_{clique(x)}$. □

3.2. Clique Tree from a PMO

According to Invariant 1, in an execution of the algorithm CliqueTree, the cliques $K_1, \ldots, K_s$ are the maximal cliques of $H(V')$ and, therefore, cliques of H that are not necessarily maximal in H. Some PEOs build the maximal cliques of H one after another: at each time in an execution of the algorithm, each clique K_j different from K_s is a maximal clique of H, and a vertex is added to K_s at each iteration of the **for** loop until K_s is a maximal clique of H; and s is incremented to start a new maximal clique of H. If $\alpha = (x_1, \ldots, x_n)$ is such a PEO, at the beginning of iteration $i < n$, the current clique K_s is equal to $N_H^{\alpha+}[x_{i+1}]$, and if K_s is not a maximal clique of H, then $K_s = S = N_H^{\alpha+}(x_i)$ and x_i is added to K_s at iteration i.

Definition 4. *A MCComp (Maximal Clique Completing) PEO of a connected chordal graph H is a PEO $\alpha = (x_1, \ldots, x_n)$ of H such that for each $i \in [1, n-1]$, $N_H^{\alpha+}[x_{i+1}]$ is a maximal clique of H or equal to $N_H^{\alpha+}(x_i)$.*

Example 1. *Let H be the graph shown in Figure 1, whose maximal cliques are $K = \{a, b, f\}$, $K' = \{c, d, e\}$ and $\{e, f\}$, and let $\alpha = (a, b, c, d, e, f)$. An execution of CliqueTree on H and α successively completes the maximal cliques $\{e, f\}$, K' and K, and we easily check that α is a MCComp PEO of H. Now, let $\beta = (a, c, d, b, e, f)$. An execution of CliqueTree on H and β successively completes $\{e, f\}$, starts K, starts and completes K' and, finally, completes K. β is not a MCComp PEO of H since for $i = 3$, $N_H^{\beta+}[x_{i+1}]$ is neither a maximal clique of H, nor equal to $N_H^{\beta+}(x_i)$ ($N_H^{\beta+}[x_{i+1}] = N_H^{\beta+}[b] = \{b, f\}$ and $N_H^{\beta+}(x_i) = N_H^{\beta+}(d) = \{e\}$).*

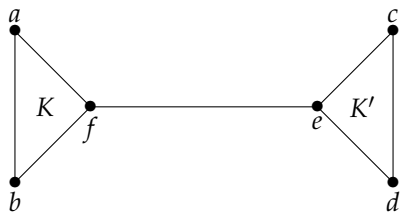

Figure 1. A chordal graph H.

Using a MCComp PEO instead of an arbitrary PEO, the algorithm CliqueTree can be simplified into the algorithm MCComp-CliqueTree (Algorithm 4) containing the blocks **InitCT**, **StartClique**, **IncreaseClique** and **DefineCT**, which will be used in further algorithms in this paper.

InitCT // *Initialize the Clique Tree*
$V' \leftarrow \emptyset$; $s \leftarrow 1$; $K_1 \leftarrow \emptyset$; $E \leftarrow \emptyset$; $Sep \leftarrow \emptyset$;

StartClique // *Start a new Clique*
$s \leftarrow s + 1$; $K_s \leftarrow S$;
$k \leftarrow min\{j, \alpha(j) \in S\}$; $p \leftarrow clique(\alpha(k))$;
$E \leftarrow E \cup \{(p, s)\}$; $Sep \leftarrow Sep \cup \{S\}$;

IncreaseClique // *Increase the current Clique*
$K_s \leftarrow K_s \cup \{x\}$; $clique(x) \leftarrow s$;

DefineCT // *Define the Clique Tree*
$T \leftarrow (\{K_1, \ldots, K_s\}, \{K_p K_q,\ (p, q) \in E\})$;

Algorithm 4: MCComp-CliqueTree.

input : a connected chordal graph H and a MCComp PEO $\alpha = (x_1, \ldots, x_n)$ of H
output: a clique tree T and the set Sep of minimal separators of H

```
InitCT;
for i = n down to 1 do
    x ← x_i; S ← N_H(x) ∩ V'; // S = N_H^{α+}(x)
    if K_s ≠ S then
        StartClique;
    end
    IncreaseClique;
    V' ← V' ∪ {x};
end
DefineCT;
```

Theorem 2. *The algorithm MCComp-CliqueTree computes a clique tree and the set of minimal separators of a connected chordal graph H from a MCComp PEO of H in linear time.*

Proof. The proof of complexity is similar to that of Theorem 1, and the correctness of the algorithm follows from Invariant 2. □

Invariant 2. *The following proposition is an invariant of the **for** loop of the algorithm MCComp-CliqueTree:*

- *(a) (b) (c) (as in Invariant 1)*
- *(d) $\forall j \in [1, s-1]$ K_j is a maximal clique of H,*
- *(e) $K_s = N_H^{\alpha+}[x_i]$.*

Proof. The proposition clearly holds at the initialization step (except for (e), which is undefined). Let us show that it is preserved at iteration i, with $1 \leq i \leq n$. Let (a1) (resp. (b1), ..., (e1)) denote Item (a) (resp. (b), ..., (e)) at the beginning of iteration i (which is supposed to be true), and let s be the value of this variable at the beginning of iteration i. We prove that (a), (b) and (c) are preserved as in the proof of Invariant 1, except that we have moreover to show that if $K_s \neq S$, then $K_p \neq S$. It is evident if $p = s$; otherwise, it follows from the fact that K_p is a maximal clique of H by (d1), whereas S is not, since it is a strict subset of the clique $S \cup \{x\}$. Hence, (a), (b) and (c) are preserved. Let us show that (d) is preserved. We only have to check that in case s is incremented (to $s+1$), K_s is a maximal clique of H. As s is incremented at iteration i $K_s \neq S$, so $i < n$, and therefore, by (e1), $K_s = N_H^{\alpha+}[x_{i+1}]$. As $K_s \neq S$ with $S = N_H^{\alpha+}(x_i)$ and α is a MCComp PEO of H, it follows that K_s is a maximal clique of H. Thus, (d) is preserved, and (e) obviously holds at the end of iteration i. □

Characterization 3. *An ordering α of a connected chordal graph H is a MCComp PEO of H if and only if it is a PMO of H.*

Proof. We prove this by induction on $n = |V|$. The result trivially holds if $n = 1$. We suppose that it holds if $|V| < n$. Let us show that it holds if $|V| = n$. Let $\alpha = (x_1, \ldots, x_n)$ be an ordering of H, and let $H' = H(V \setminus \{x_1\})$.

$\Rightarrow$: We suppose that α is a MCComp PEO of H. Let us show that it is a PMO of H. $(x_2, \ldots, x_n)$ is a MCComp PEO of H', so by the induction hypothesis, it is a PMO of H' compatible with a perfect moplex partition of H', say $(X_1, \ldots, X_k)$. If $N_H^{\alpha+}(x_1) = N_H^{\alpha+}[x_2]$, then $(\{x_1\} \cup X_1, \ldots, X_k)$ is a perfect moplex partition of H. Otherwise, in an execution of the algorithm MCComp-CliqueTree on H and α, $K_s \neq S$ at Iteration 1 since $S = N_H^{\alpha+}(x_1)$ and $K_s = N_H^{\alpha+}[x_2]$ by Invariant 2 (e), so S is a minimal separator of H, which makes $\{x_1\}$ a moplex of H and $(\{x_1\}, X_1, \ldots, X_k)$ a perfect moplex partition of H. Hence, α is a PMO of H.

$\Leftarrow$: We suppose that α is a PMO of H compatible with perfect moplex partition $(X_1, \ldots, X_k)$. Let us show that it is a MCComp PEO of H. As it is a PMO of H, it is a PEO of H. $(x_2, \ldots, x_n)$ is a PMO of H' (compatible with perfect moplex partition $(X_2, \ldots, X_k)$ if $X_1 = \{x_1\}$ and $(X_1 \setminus \{x_1\}, \ldots, X_k)$ otherwise), so by the induction hypothesis, it is a MCComp PEO of H'. Hence, for each i from two to $n-1$, $N_H^{\alpha+}[x_{i+1}]$ is a maximal clique of H' or equal to $N_H^{\alpha+}(x_i)$. It is sufficient to show that for each i from two to $n-1$, if $N_H^{\alpha+}[x_{i+1}]$ is a maximal clique of H', then it is a maximal clique of H and that $N_H^{\alpha+}[x_2]$ is a maximal clique of H or equal to $N_H^{\alpha+}(x_1)$.

First case: $X_1 = \{x_1\}$

As $N_H(x_1)$ is a minimal separator of H, by Characterization 1, it is equal to the intersection of two maximal cliques of H and, therefore, is not a maximal clique of H'. It follows by Lemma 2 that each maximal clique of H' is a maximal clique of H. Moreover, $N_H^{\alpha+}[x_2] = N_{H'}[X_2]$, which is a maximal clique of H' and therefore of H.

Second case: $X_1 \neq \{x_1\}$

In that case, $N_H^{\alpha+}(x_1) = N_H^{\alpha+}[x_2]$. By Lemma 2, each maximal clique of H' different from $N_H(x_1)$ is a maximal clique of H. It follows that for each i from two to $n-1$, if $N_H^{\alpha+}[x_{i+1}]$ is a maximal clique of H', then it is a maximal clique of H, since it does not contain x_2, whereas $N_H(x_1)$ contains x_2. □

Corollary 1. *The algorithm MCComp-CliqueTree computes a clique tree and the set of minimal separators of a connected chordal graph H from a PMO of H in linear time.*

3.3. Clique Tree Using MLS

The algorithms MCS, LexBFS, LexDFS and, more generally, the algorithm $\mathcal{L}$-MLS for any labeling structure $\mathcal{L}$ for which the order on labels is total compute a PMO of a connected chordal graph [12]. Note that the definition of a labeling structure given in [12] is less general than the definition given in this paper, but the proof of this result still holds here. We define the algorithm moplex-MLS (Algorithm 5), which computes a PMO of a chordal graph, whether the order on labels is total or not, by adding, in the case where the ordering fails to be total, a tie-breaking rule for choosing a vertex with a maximal label.

Algorithm 5: moplex-MLS.

input : a connected chordal graph H and a labeling structure $\mathcal{L} = (L, \preceq, l_0, Inc)$
output: a PMO $\alpha = (x_1, \ldots, x_n)$ of H

$V' \leftarrow \emptyset$; initialize all labels as l_0; *prev-max-label* $\leftarrow l_0$;
for $i = n$ ***down to*** 1 **do**
  Choose a vertex x in $V \setminus V'$ whose label is maximal,
  and if possible strictly greater than *prev-max-label*;
  $x_i \leftarrow x$;
  foreach y *in* $N_H(x) \setminus V'$ **do**
    $label(y) \leftarrow Inc(label(y), i)$;
  end
  $V' \leftarrow V' \cup \{x\}$; *prev-max-label* $\leftarrow label(x)$;
end

Theorem 3. *The algorithm moplex-MLS computes a PMO of a connected chordal graph.*

To prove Theorem 3, we will use the following Lemma.

Lemma 3. *In an execution of moplex-MLS, for each i in $[1, n-1]$ and each y in V such that $\alpha^{-1}(y) \leq i$, at the beginning of iteration i,*

(a) If $N_H^{\alpha+}[x_{i+1}] \subseteq N_H^{\alpha,i}(y)$, then prev-max-label $\prec label(y)$.
(b) If prev-max-label $\prec label(y)$, then $\{x_{i+1}\} \subseteq N_H^{\alpha,i}(y) \subseteq N_H^{\alpha+}[x_{i+1}]$.

Proof. (a) If $N_H^{\alpha+}[x_{i+1}] \subseteq N_H^{\alpha,i}(y)$, then $N_H^{\alpha,i}(x_{i+1}) \subset N_H^{\alpha,i}(y)$, so by Lemma 1, *prev-max-label* $= label(x_{i+1}) \prec label(y)$.

(b) We suppose that *prev-max-label* $\prec label(y)$. As the label of x_{i+1} is maximal at the beginning of iteration $i+1$, $label(y)$ has been increased during iteration $i+1$, so y is a neighbor of x_{i+1} in H. As α is an MLS ordering of H, it is a PEO of H, so $N_H^{\alpha+}(y)$ is a clique containing x_{i+1}, and therefore, $N_H^{\alpha,i}(y) \subseteq N_H^{\alpha+}[x_{i+1}]$. □

Proof. (of Theorem 3) Let $\alpha = (x_1, \ldots, x_n)$ be the ordering computed by an execution of moplex-MLS on input H and $\mathcal{L}$. Let us show that it is a PMO of H. By Characterization 3, it is sufficient to show that α is a MCComp PEO of H. As α is an MLS ordering of H, it is a PEO of H. Let $i \in [1, n-1]$.

We suppose that $N_H^{\alpha+}[x_{i+1}]$ is not a maximal clique of H. Let us show that it is equal to $N_H^{\alpha+}(x_i)$. As $N_H^{\alpha+}[x_{i+1}]$ is not a maximal clique of H, there is a vertex y, such that $\alpha^{-1}(y) \leq i$ and $N_H^{\alpha+}[x_{i+1}] \subseteq N_H^{\alpha,i}(y)$, and therefore, by Lemma 3 (a), *prev-max-label* $\prec$ $label(y)$ at the beginning of iteration i in this execution. It follows by the condition on the choice of x that *prev-max-label* $\prec$ $label(x_i)$, and therefore, by Lemma 3 (b), $N_H^{\alpha+}(x_i) \subseteq N_H^{\alpha+}[x_{i+1}]$. It is impossible that $N_H^{\alpha+}(x_i) \subset N_H^{\alpha+}[x_{i+1}]$ since in that case $N_H^{\alpha,i}(x_i) \subset N_H^{\alpha,i}(y)$, so by Lemma 1 at the beginning of iteration i, $label(x_i) \prec label(y)$ and x_i would not be a vertex with maximal label. Hence, $N_H^{\alpha+}(x_i) = N_H^{\alpha+}[x_{i+1}]$. □

If $\mathcal{L}$ is a labeling structure with a total order on labels, condition "if possible strictly greater than *prev-max-label*" is useless, so the algorithm moplex-$\mathcal{L}$-MLS is actually identical to $\mathcal{L}$-MLS. We thus provide an alternate proof for the result from [12] that if the order on labels is total, then each $\mathcal{L}$-MLS ordering of a chordal graph is a PMO of this graph, with a more general definition of a labeling structure and an alternative (simpler) proof.

Combining algorithms MCComp-CliqueTree and moplex-MLS, we define the algorithm MLS-CliqueTree (Algorithm 6) computing both a PMO and a clique tree of a chordal graph.

Algorithm 6: MLS-CliqueTree.

input : a connected chordal graph H and a labeling structure $\mathcal{L} = (L, \preceq, l_0, Inc)$
output: a PMO $\alpha = (x_1, \ldots, x_n)$, a clique tree T and the set Sep of minimal separators of H

InitCT;
Initialize all labels as l_0; *prev-max-label* $\leftarrow l_0$;
for $i = n$ ***down to*** 1 **do**
  Choose a vertex x in $V \setminus V'$ whose label is maximal,
  and if possible strictly greater than *prev-max-label*;
  $x_i \leftarrow x$; $S \leftarrow N_H(x) \cap V'$; // $S = N_H^{\alpha+}(x)$
  if $K_s \neq S$ **then**
    StartClique;
  end
  IncreaseClique;
  foreach y *in* $N_H(x) \setminus V'$ **do**
    $label(y) \leftarrow Inc(label(y), i)$;
  end
  $V' \leftarrow V' \cup \{x\}$; *prev-max-label* $\leftarrow label(x)$;
end
DefineCT;

The correctness of the algorithm MLS-CliqueTree immediately follows from Corollary 1 and Theorem 3.

Example 2. *Consider an execution of the algorithm MLS-CliqueTree on the graph H shown in Figure 1 and labeling structure $\mathcal{L}_X$ with $X \in \{MCS, LexBFS, LexDFS, MNS\}$ choosing vertices f, then e first (and, therefore, completing the maximal clique $\{e, f\}$ first). Then, the execution successively completes K, then K′ if $X = LexBFS$, K′, then K if $X = LexDFS$, and either K, then K′ or K′, then K otherwise. Removing condition "if possible strictly greater than prev-max-label" has no effect if $X \neq MNS$, but if $X = MNS$, it would allow the execution to choose alternatively a vertex of K and a vertex of K′, as the labels of the vertices of K are incomparable to the labels of the vertices of K′.*

The algorithm MLS-CliqueTree generalizes the algorithm extended-MCS from [4] and its extension to LexBFS from [13], except that in these algorithms, the condition "$K_s \neq S$" is replaced by a direct condition on labels: "$label(x) \preceq$ *prev-max-label*". We define a necessary and sufficient

condition on a labeling structure $\mathcal{L}$ for the replacement of "$K_s \neq S$" by "*prev-max-label* $\not\prec$ $label(x)$" (which becomes "$label(x) \preceq$ *prev-max-label*" if $\preceq$ is a total order) to be possible.

Definition 5. *Let $\mathcal{L} = (L, \preceq, l_0, Inc)$ be a labeling structure. $\mathcal{L}$ is DCL (Detect new Cliques with Labels) if for any integers i and n such that $1 \leq i < n$ and any subsets I and I' of $[i+2, n]$, if $I \subseteq I'$ and $lab_{\mathcal{L}}(I') \prec lab_{\mathcal{L}}(I \cup \{i+1\})$, then $I = I'$.*

The labeling structures associated with MCS, LexBFS and MNS are clearly DCL, but $\mathcal{L}_{LexDFS}$ is not since for any subsets I and I' of $[i+2, n]$, $lab_{\mathcal{L}}(I') \prec lab_{\mathcal{L}}(I \cup \{i+1\})$ necessarily holds.

Remark 1. *For each $X \in \{MCS, LexBFS, MNS\}$, $\mathcal{L}_X$ is a DCL labeling structure, but $\mathcal{L}_{LexDFS}$ is not.*

On a DCL labeling structure, the algorithm DCL-MLS-CliqueTree (Algorithm 7) detects new cliques by a condition on abels.

Algorithm 7: DCL-MLS-CliqueTree.

input : a connected chordal graph H and a DCL labeling structure $\mathcal{L} = (L, \preceq, l_0, Inc)$
output: a PMO $\alpha = (x_1, \ldots, x_n)$, a clique tree T and the set Sep of minimal separators of H

```
InitCT;
Initialize all labels as l_0; prev-max-label ← l_0;
for i = n down to 1 do
    Choose a vertex x in V \ V' whose label is maximal,
    and if possible strictly greater than prev-max-label;
    x_i ← x; S ← N_H(x) ∩ V'; // S = N_H^{α+}(x)
    if prev-max-label ⊀ label(x) and i < n then
        StartClique;
    end
    IncreaseClique;
    foreach y in N_H(x) \ V' do
        label(y) ← Inc(label(y), i);
    end
    V' ← V' ∪ {x}; prev-max-label ← label(x);
end
DefineCT;
```

Theorem 4. *The algorithm DCL-MLS-CliqueTree is correct and would be incorrect with any non-DCL input labeling structure. Moreover, if the input labeling structure is $\mathcal{L}_X$ with $X \in \{MCS, LexBFS\}$, then the algorithm runs in linear time.*

Proof. We suppose that $\mathcal{L}$ is DCL. It is sufficient to show that at each iteration i in $[1, n-1]$, $K_s = S \Leftrightarrow$ *prev-max-label* $\prec label(x)$, i.e., by Invariant 2 (e), $N_H^{\alpha+}[x_{i+1}] = N_H^{\alpha,i}(x_i) \Leftrightarrow$ *prev-max-label* $\prec label(x)$. The implication from left to right immediately follows from Lemma 3 (a). Let us show the reverse implication. We suppose that *prev-max-label* $\prec label(x_i)$. By Lemma 3 (b), $\{x_{i+1}\} \subseteq N_H^{\alpha,i}(x_i) \subseteq N_H^{\alpha+}[x_{i+1}]$. Let $I = \alpha^{-1}(N_H^{\alpha,i+1}(x_i))$, and let $I' = \alpha^{-1}(N_H^{\alpha+}(x_{i+1}))$. $I \subseteq I' \subseteq [i+2, n]$ and $lab_{\mathcal{L}}(I') =$ *prev-max-label* $\prec label(x_i) = lab_{\mathcal{L}}(I \cup \{i+1\})$, so $I = I'$, since $\mathcal{L}$ is DCL. It follows that $N_H^{\alpha+}[x_{i+1}] = N_H^{\alpha,i}(x_i)$.

We suppose now that $\mathcal{L}$ is not DCL. Then, there are some integers i and n with $1 \leq i < n$ and some subsets I and I' of $[i+2, n]$ such that $I \subset I'$ and $lab_{\mathcal{L}}(I') \prec lab_{\mathcal{L}}(I \cup \{i+1\})$. Let $H = (V, E)$ and α be defined by: $V = \{1, \ldots, n\}$, $\alpha = (1, \ldots, n)$, $\{i+2, \ldots, n\}$ is a clique of H, $N_H^{\alpha+}(i+1) = I'$ and $\forall j \in [1, i]$ $N_H^{\alpha+}(j) = I \cup \{i+1\}$. H is connected and chordal and by IC α can be computed by

an execution of the algorithm DCL-MLS-CliqueTree. At iteration i of such an execution, $K_s \neq S$, but $\textit{prev-max-label} = lab_{\mathcal{L}}(I') \prec lab_{\mathcal{L}}(I \cup \{i+1\}) = label(x)$, so the execution increases current clique K_s instead of starting a new one.

We suppose that the input labeling structure is $\mathcal{L}_X$ with $X \in \{MCS, LexBFS\}$. As the order on labels is total, it is sufficient to choose a vertex x with the maximal label at each iteration. As $\mathcal{L}_X$-MLS runs in linear time, it is sufficient to check that condition $\textit{prev-max-label} \not\prec label(x)$ can be evaluated in $O(|N_H(x)|)$ time. It is obviously the case if $X = MCS$. It is also the case if $X = LexBFS$ since $label(x)$ is of a length of at most $|N_H(x)|$. □

For $X = MCS$ (resp. $LexBFS$), as $\mathcal{L}_X$ is DCL with totally ordered labels, the algorithm DCL-$\mathcal{L}_X$-MLS-CliqueTree can be simplified by choosing an arbitrary vertex with the maximal label at each iteration, yielding the algorithms from [4] (resp. [13]). As $\mathcal{L}_{LexDFS}$ is not DCL, it follows from Theorem 4 that the algorithm DCL-MLS-CliqueTree would be incorrect with $\mathcal{L}_{LexDFS}$ as the input labeling structure. Note that this contradicts Theorem 4.1 from [14] stating that in an execution of LexDFS, $label(x_i) \preceq \textit{prev-max-label}$ is a necessary and sufficient condition for $N_H^{\alpha+}[x_{i+1}]$ to be a maximal clique and $N_H^{\alpha+}(x_i)$ to be a minimal separator of the input graph, implying that DCL-MLS-CliqueTree is correct with $\mathcal{L}_{LexDFS}$ as the input labeling structure. The simple graph H from Figure 1 is a counterexample as shown below.

Counterexample 1. *An execution of the algorithm DCL-MLS-CliqueTree on the graph H shown in Figure 1 and the labeling structure $\mathcal{L}_{LexDFS}$ computing ordering (a, b, c, d, e, f) is shown in Figure 2. For each vertex x, the number $\alpha^{-1}(x)$ and the final label of x are indicated. At the beginning of Iteration 4, $label(a) = label(b) = \textit{pre-max-label} = (6)$ and $label(c) = label(d) = (5)$, with $(6) \prec (5)$ according to labeling structure $\mathcal{L}_{LexDFS}$. At Iteration 4, vertex d is chosen, and as $\textit{prev-max-label} \prec label(d)$, the execution increases the current clique $\{e, f\}$ instead of starting new clique K'.*

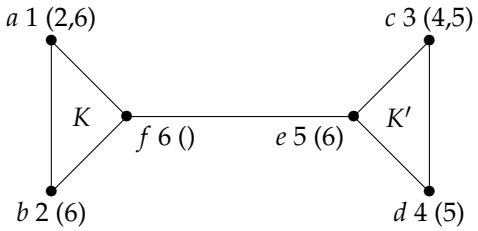

Figure 2. LexDFS labels do not detect new maximal cliques.

4. Clique Tree of the Complement Graph

The algorithm MLS can be used to compute a PEO of the complement graph [11]. We will show that it can be used to compute a PMO, a clique tree and the minimal separators of the complement graph, provided that this complement graph is connected and chordal.

Definition 6. *Let $\mathcal{L} = (L, \preceq, l_0, Inc)$ be a labeling structure. $\mathcal{L}$ is complement-reversing if for any integers i and n with $1 \leq i \leq n$ and any subsets I and I' of $[i, n]$, if $lab_{\mathcal{L}}(I) \preceq lab_{\mathcal{L}}(I')$, then $lab_{\mathcal{L}}([i, n] \setminus I') \preceq lab_{\mathcal{L}}([i, n] \setminus I)$.*

Remark 2. *[11] For each $X \in \{MCS, LexBFS, LexDFS, MNS\}$, $\mathcal{L}_X$ is complement-reversing.*

Definition 7. *For any labeling structure $\mathcal{L}$, $Rev(\mathcal{L})$ denotes the labeling structure obtained from $\mathcal{L}$ by replacing the order on the labels by its dual order.*

Theorem 5. *[11] Let $\mathcal{L}$ be a complement-reversing labeling structure. Then, an ordering of a graph G is a $\mathcal{L}$-MLS ordering of $\overline{G}$ if and only if it is a $Rev(\mathcal{L})$-MLS ordering of G.*

Thus, if $\mathcal{L}$ is complement-reversing, then replacing "maximal" by "minimal" in the algorithm MLS results in an algorithm that, with G and $\mathcal{L}$ as input, computes a $Rev(\mathcal{L})$-MLS ordering of G and therefore a $\mathcal{L}$-MLS ordering of $\overline{G}$, which is a PEO of $\overline{G}$ if it is chordal. However, it is not correct to replace in the algorithm moplex-MLS or MLS-CliqueTree condition "if possible strictly greater than *prev-max-label*" by "if possible strictly smaller than *prev-max-label*" since in the case $N_H^{\alpha+}[x_{i+1}] = N_H^{\alpha+}(x_i)$. With $H = \overline{G}$, we have $N_G^{\alpha+}(x_{i+1}) = N_G^{\alpha+}(x_i)$, i.e., *prev-min-label* $= label(x)$ at iteration i (as x is adjacent to x_{i+1} in H, it is not adjacent to x_{i+1} in G, and therefore, its label is not increased during iteration $i+1$). We will show that *prev-min-label* $\neq label(x)$ is a necessary and sufficient condition for starting a new clique in H, whether the labeling structure is DCL or not.

Thus, the algorithm complement-DCL-MLS-CliqueTree (Algorithm 8) detects new cliques on the complement of the input graph by a condition on labels whether the labeling structure is DCL or not.

Algorithm 8: complement-DCL-MLS-CliqueTree.

input : a graph G whose complement is a connected chordal graph and a complement-reversing labeling structure $\mathcal{L} = (L, \preceq, l_0, Inc)$
output: a PMO $\alpha = (x_1, \ldots, x_n)$, a clique tree T and the set Sep of minimal separators of $\overline{G}$

$H \leftarrow \overline{G}$; **InitCT**;
Initialize all labels as l_0; *prev-min-label* $\leftarrow l_0$;
for $i = n$ ***down to*** 1 **do**
  Choose a vertex x in $V \setminus V'$ whose label is minimal, and if possible equal to *prev-min-label*;
  $x_i \leftarrow x$; $S \leftarrow N_H(x) \cap V'$; // $S = N_H^{\alpha+}(x)$
  if *prev-min-label* $\neq label(x)$ *and* $i < n$ **then**
    StartClique; *prev-min-label* $\leftarrow label(x)$;
  end
  IncreaseClique;
  foreach y *in* $N_G(x) \setminus V'$ **do**
    $label(y) \leftarrow Inc(label(y), i)$;
  end
  $V' \leftarrow V' \cup \{x\}$;
end
DefineCT;

Note that as by condition, IC labels can only increase, *prev-min-label* is necessarily minimal.

Theorem 6. *The algorithm complement-DCL-MLS-CliqueTree is correct.*

An ordering computed by the algorithm complement-DCL-MLS-CliqueTree is a $Rev(\mathcal{L})$-MLS ordering of G, and therefore, by Theorem 5 a $\mathcal{L}$-MLS ordering of H, which is a PEO of H. To prove Theorem 6, we will use the following Lemma.

Lemma 4. *In an execution of complement-DCL-MLS-CliqueTree, for each i in $[1, n-1]$ and each y in V such that $\alpha^{-1}(y) \leq i$, at the beginning of iteration i, the following propositions are equivalent:*

(1) $N_H^{\alpha+}[x_{i+1}] \subseteq N_H^{\alpha,i}(y)$,
(2) *prev-min-label* $= label(y)$,
(3) $N_H^{\alpha+}[x_{i+1}] = N_H^{\alpha,i}(y)$.

Proof. (1) ⇒ (2): We suppose that $N_H^{\alpha+}[x_{i+1}] \subseteq N_H^{\alpha,i}(y)$. Then, $N_G^{\alpha,i}(y) \subseteq N_G^{\alpha,i}(x_{i+1})$, and therefore, by Lemma 1, $label(y) \preceq label(x_{i+1}) = prev\text{-}min\text{-}label$. Moreover, $label(y) \neq prev\text{-}min\text{-}label$ since $label(y)$ and *prev-min-label* are the labels of y and x_{i+1}, respectively, at the beginning of iteration $i+1$. Hence, $prev\text{-}min\text{-}label = label(y)$.

(2) ⇒ (3): We suppose that $prev\text{-}min\text{-}label = label(y)$. Then, the label of y is not increased during iteration $i+1$ (otherwise, it would have been strictly smaller than the label of x_{i+1} at the beginning of iteration $i+1$). It follows that y is not adjacent to x_{i+1} in G and therefore is adjacent to x_{i+1} in H. As α is a PEO of H, $N_H^{\alpha,i}(y) \subseteq N_H^{\alpha+}[x_{i+1}]$. Moreover, $N_H^{\alpha,i}(y) \not\subset N_H^{\alpha+}[x_{i+1}]$, since otherwise, $N_G^{\alpha,i}(x_{i+1}) \subset N_G^{\alpha,i}(y)$, and therefore, by Lemma 1, $prev\text{-}min\text{-}label \prec label(y)$. Hence, $N_H^{\alpha+}[x_{i+1}] = N_H^{\alpha,i}(y)$.

(3) ⇒ (1) is evident. □

Proof. (of Theorem 6) Let $\alpha = (x_1, \ldots, x_n)$ be the ordering computed by an execution of complement-DCL-MLS-CliqueTree on input G and $\mathcal{L}$, and let $H = \overline{G}$. Let us show that it is a PMO of H, i.e., a MCComp PEO of H by Characterization 3. As α is an MLS ordering of H, it is a PEO of H. Let $i \in [1, n-1]$. We suppose that $N_H^{\alpha+}[x_{i+1}]$ is not a maximal clique of H. Let us show that it is equal to $N_H^{\alpha+}(x_i)$. As $N_H^{\alpha+}[x_{i+1}]$ is not a maximal clique of H, there is a vertex y, such that $\alpha^{-1}(y) \leq i$ and $N_H^{\alpha+}[x_{i+1}] \subseteq N_H^{\alpha,i}(y)$, and therefore, by Lemma 4, $prev\text{-}min\text{-}label = label(y)$ at the beginning of iteration i in this execution. It follows by the condition on the choice of x that $prev\text{-}min\text{-}label = label(x_i)$ and, therefore, by Lemma 4 that $N_H^{\alpha+}[x_{i+1}] = N_H^{\alpha+}(x_i)$.

It remains to show that condition $prev\text{-}min\text{-}label \neq label(x)$ correctly detects new cliques. It is evident at iteration n. For each iteration i with $i < n$, it immediately follows from Lemma 4, as the condition to start a new clique is $K_s \neq S$, i.e., $N_H^{\alpha+}[x_{i+1}] \neq N_H^{\alpha,i}(x)$ by Invariant 2 (e). □

Example 3. *Let G be the complement graph of the graph H shown in Figure 1. An execution of the algorithm complement-DCL-MLS-CliqueTree on G and labeling structure $\mathcal{L}_{LexDFS}$ computing ordering (a, b, c, d, e, f) is shown in Figure 3. For each vertex x, the number $\alpha^{-1}(x)$ and the final label of x are indicated. At the beginning of Iteration 4, $label(a) = label(b) = (5)$ and $label(c) = label(d) = (6)$, with $(6) \prec (5)$ according to labeling structure $\mathcal{L}_{LexDFS}$ and $pre\text{-}min\text{-}label = ()$; vertex d is chosen, and new clique K' is started at Iteration 4 as desired. Clique K is correctly started at Iteration 2 since $pre\text{-}min\text{-}label = (6)$ and $label(x) = label(b) = (3, 4, 5)$.*

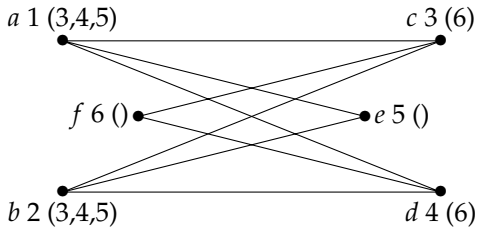

Figure 3. Lexicographic Depth-First Search (LexDFS) labels detect new cliques on the complement.

The algorithm complement-DCL-MLS-CliqueTree does not run in $O(n+m)$ time, since computing S (in order to compute the maximal cliques and minimal separators of $\overline{G}$) and k in StartClique (in order to compute the edges of the clique tree) globally, takes $O(n+\overline{m})$ time, where $\overline{m}$ is the number of edges of $\overline{G}$. However, the algorithm complement-DCL-MLS-generators (Algorithm 9) computes a PMO α and the generators of the maximal cliques and minimal separators w.r.t. α of $\overline{G}$ in $O(n+m)$ time.

Algorithm 9: complement-DCL-MLS-generators.

input : a graph G whose complement is a connected chordal graph and a complement-reversing labeling structure $\mathcal{L} = (L, \preceq, l_0, Inc)$
output: a PMO $\alpha = (x_1, \ldots, x_n)$ and the sets *GenCli* and *GenSep* of generators of the maximal cliques and minimal separators w.r.t. α of $\overline{G}$ respectively

$V' \leftarrow \emptyset$; $GenCli \leftarrow \emptyset$;$GenSep \leftarrow \emptyset$;
Initialize all labels as l_0; *prev-min-label* $\leftarrow l_0$;
for $i = n$ ***down to*** 1 **do**
  Choose a vertex x in $V \setminus V'$ whose label is minimal, and if possible equal to *prev-min-label*;
  $x_i \leftarrow x$;
  if *prev-min-label* $\neq label(x)$ *and* $i < n$ **then**
    $GenCli \leftarrow GenCli + \{x_{i+1}\}$; $GenSep \leftarrow GenSep + \{x_i\}$;
    prev-min-label $\leftarrow label(x)$;
  end
  foreach y *in* $N_G(x) \setminus V'$ **do**
    $label(y) \leftarrow Inc(label(y), i)$;
  end
  $V' \leftarrow V' \cup \{x\}$;
end
$GenCli \leftarrow GenCli + \{x_1\}$;

Theorem 7. *The algorithm complement-DCL-MLS-generators is correct, and if the input labeling structure is $\mathcal{L}_X$ with $X \in \{MCS, LexBFS\}$, then it runs in linear time.*

Proof. Correctness follows from the correctness of complement-DCL-MLS-CliqueTree.

We suppose that the input labeling structure is $\mathcal{L}_X$ with $X \in \{MCS, LexBFS\}$. As the order on the labels is total, it is sufficient to choose a vertex x with the minimal label at each iteration. The linear time complexity of $\mathcal{L}_X$-MLS also holds for $Rev(\mathcal{L}_X)$-MLS. Hence, it is sufficient to check that condition *prev-min-label* $\neq label(x)$ can be evaluated in $O(|N_G(x)|)$ time. This is obviously the case if $X = MCS$. This is also the case if $X = LexBFS$, since $label(x)$ is of length at most $|N_G(x)|$. □

5. Extended Results

5.1. Clique Tree of a Minimal Triangulation

The algorithm MLSM computes an MEO and the associated minimal triangulation of the input graph G. It computes an MMO of G if the order on labels is total [12]. It can be modified into the algorithm moplex-MLSM computing an MMO of G whether the order on labels is total or not, which can be extended to the algorithms MLSM-CliqueTree and DCL-MLSM-CliqueTree computing an MMO, the associated minimal triangulation H of G, a clique tree and the minimal separators of H. Below are the algorithms moplex-MLSM and DCL-MLSM-CliqueTree (Algorithms 10 and 11).

Algorithm 10: moplex-MLSM.

input : a connected graph G and a labeling structure $\mathcal{L} = (L, \preceq, l_0, Inc)$
output: an MMO $\alpha = (x_1, \ldots, x_n)$ of G and the associated minimal triangulation $H = G^+_\alpha$

$H \leftarrow G$; $V' \leftarrow \emptyset$; initialize all labels as l_0; *prev-max-label* $\leftarrow l_0$;
for $i = n$ ***down to*** 1 **do**
  Choose a vertex x in $V \setminus V'$ whose label is maximal,
  and if possible strictly greater than *prev-max-label*;
  $x_i \leftarrow x$;
  $Y \leftarrow \emptyset$;
  foreach y *in* $V \setminus V'$ **do**
    if *there is a path* μ *of length* ≥ 1 *in* $G(V \setminus V')$ *between* x *and* y *such that for each internal vertex* z *of* μ $label(z) \prec label(y)$ **then**
      $Y \leftarrow Y + \{y\}$; $E(H) \leftarrow E(H) \cup \{xy\}$;
    end
  end
  foreach y *in* Y **do**
    $label(y) \leftarrow Inc(label(y), i)$;
  end
  $V' \leftarrow V' \cup \{x\}$; *prev-max-label* $\leftarrow label(x)$;
end

Theorem 8. *The algorithm moplex-MLSM computes an MMO and the associated minimal triangulation of the input graph.*

To prove Theorem 8, we will use the following lemmas.

Lemma 5. *[8] A moplex of a minimal triangulation of G is a moplex of G.*

Lemma 6. *If α is an MEO of G and a PMO of G^+_α, then it is an MMO of G.*

Proof. Let $H = G^+_\alpha$. We prove this by induction on the size k of the perfect moplex partition $(X_1, \ldots, X_k)$ associated with α in H. If $k = 1$, then H is a clique, so G is a clique, as well, since H is a minimal triangulation of G, and we are done. We assume that the property holds for a perfect moplex partition of size $k \geq 1$. Let $(X_1, \ldots, X_{k+1})$ be the perfect moplex partition associated with α in H. As X_1 is a moplex of H, by Lemma 5, it is a moplex of G. Let G_1 be the graph obtained from G by saturating $N_G(X_1)$ and removing X_1, and let α_1 be the restriction of α to $V \setminus X_1$.

As α is an MEO of G, α_1 is an MEO of G_1 and as $(G_1)^+_{\alpha_1} = H(V \setminus X_1)$, α_1 is a PMO of $(G_1)^+_{\alpha_1}$. Hence, by the induction hypothesis, α_1 is an MMO of G_1, and therefore, α is an MMO of G. □

Note that the fact that α is a PMO of G^+_α does not imply that it is an MMO of G. For instance, if G is a non-clique graph with a universal vertex, then any ordering α of G such that $\alpha(1)$ is universal is a PMO of G^+_α (since G^+_α is a clique), but not an MMO of G (since it is not an MEO of G).

Proof. (of Theorem 8) Let α be the ordering and H be the graph computed by an execution of moplex-MLSM on input graph G. As this execution is also an execution of MLSM, α is an MEO of G and $H = G^+_\alpha$. As moreover at each iteration, the labels are increased exactly in the same way as in an execution of moplex-MLS on H, α is a PMO of H and, therefore, an MMO of G by Lemma 6. □

Algorithm 11: DCL-MLSM-CliqueTree.

input : a connected graph G and a DCL labeling structure $\mathcal{L} = (L, \preceq, l_0, Inc)$
output: an MMO $\alpha = (x_1, \ldots, x_n)$ of G, the associated minimal triangulation $H = G_\alpha^+$, a clique tree T and the set Sep of minimal separators of H

$H \leftarrow G$; **InitCT**;
Initialize all labels as l_0; *prev-max-label* $\leftarrow l_0$;
for $i = n$ ***down to*** 1 **do**
  Choose a vertex x in $V \setminus V'$ whose label is maximal,
  and if possible strictly greater than *prev-max-label*;
  $x_i \leftarrow x$; $S \leftarrow N_H(x) \cap V'$; // $S = N_H^{\alpha+}(x)$
  if *prev-max-label* $\not\prec$ $label(x)$ *and* $i < n$ **then**
    StartClique;
  end
  IncreaseClique;
  $Y \leftarrow \emptyset$;
  foreach y *in* $V \setminus V'$ **do**
    if *there is a path* μ *of length* ≥ 1 *in* $G(V \setminus V')$ *between* x *and* y *such that for each internal vertex* z *of* μ $label(z) \prec label(y)$ **then**
      $Y \leftarrow Y + \{y\}$; $E(H) \leftarrow E(H) \cup \{xy\}$;
    end
  end
  foreach y *in* Y **do**
    $label(y) \leftarrow Inc(label(y), i)$;
  end
  $V' \leftarrow V' \cup \{x\}$; *prev-max-label* $\leftarrow label(x)$;
end
DefineCT;

The correctness of the algorithm DCL-MLSM-CliqueTree immediately follows from the correctness of the algorithms moplex-MLSM and DCL-MLS-CliqueTree.

5.2. Atom Tree and Clique Minimal Separators

An atom tree of a connected graph G can be computed from a clique tree of a minimal triangulation of G as described in the following theorem.

Theorem 9. *[20] Let G be a connected graph; let H be a minimal triangulation of G; let $T = (V_T, E_T)$ be a clique tree of H; and let T' be the forest obtained from T by removing all edges KK' such that $K \cap K'$ is a clique in G; let T'' be the tree obtained from T by merging the nodes of each tree of T' into one node; then, T'' is an atom tree of G, and for each edge KK' of T such that $K \cap K'$ is a clique in G, $K \cap K' = A \cap A'$, where A and A' are the atoms of G containing K and K', respectively.*

Thus, the algorithm DCL-MLSM-CliqueTree can be modified into the algorithm DCL-MLSM-AtomTree computing an atom tree and the clique minimal separators of the input graph G: in case a new clique is started, a new atom is started only if S is a clique in G; otherwise, the atom containing K_p is increased. Note that the atoms are not built one after another, since an atom different from A_s may be increased if a new clique is started and S is not a clique in G. Variable q contains the index of the current atom. The algorithm DCL-MLSM-AtomTree (Algorithm 12) generalizes the algorithm MCSM-atom-tree from [20], while correcting an error in this algorithm (a confusion between variables q and s).

InitAT // *Initialize the Atom Tree*
$V' \leftarrow \emptyset$; $s \leftarrow 1$; $q \leftarrow 1$; $A_1 \leftarrow \emptyset$; $E \leftarrow \emptyset$; $CliqueSep \leftarrow \emptyset$;

StartAtom // *Start a new Atom*
$s \leftarrow s+1$; $A_s \leftarrow S$;
$E \leftarrow E \cup \{(p,s)\}$; $CliqueSep \leftarrow CliqueSep \cup \{S\}$;

IncreaseAtom // *Increase the current Atom*
$A_q \leftarrow A_q \cup \{x\}$; $atom(x) \leftarrow q$;

DefineAT // *Define the Atom Tree*
$T \leftarrow (\{A_1, \ldots, A_s\}, \{A_p A_q,\ (p,q) \in E\})$;

Algorithm 12: DCL-MLSM-AtomTree.

input : a connected graph G and a DCL labeling structure $\mathcal{L} = (L, \preceq, l_0, Inc)$
output: an atom tree T and the set $CliqueSep$ of clique minimal separators of G

$H \leftarrow G$; **InitAT**;
Initialize all labels as l_0; *prev-max-label* $\leftarrow l_0$;
for $i = n$ ***down to*** 1 **do**
  Choose a vertex x in $V \setminus V'$ whose label is maximal,
  and if possible strictly greater than *prev-max-label*;
  $x_i \leftarrow x$; $S \leftarrow N_H(x) \cap V'$; // $S = N_H^{\alpha+}(x)$
  if *prev-max-label* $\not\prec label(x)$ *and* $i < n$ **then**
    $k \leftarrow min\{j, \alpha(j) \in S\}$; $p \leftarrow clique(\alpha(k))$;
    if *S is a clique in G* **then**
      StartAtom; $q \leftarrow s$;
    end
    else
      $q \leftarrow p$;
    end
  end
  IncreaseAtom;
  $Y \leftarrow \emptyset$;
  foreach *y in* $V \setminus V'$ **do**
    if *there is a path* μ *of length* ≥ 1 *in* $G(V \setminus V')$ *between x and y such that for each internal vertex z of* μ $label(z) \prec label(y)$ **then**
      $Y \leftarrow Y + \{y\}$; $E(H) \leftarrow E(H) \cup \{xy\}$;
    end
  end
  foreach *y in* Y **do**
    $label(y) \leftarrow Inc(label(y), i)$;
  end
  $V' \leftarrow V' \cup \{x\}$; *prev-max-label* $\leftarrow label(x)$;
end
DefineAT;

The correctness of the algorithm DCL-MLSM-AtomTree follows from Theorem 9, Characterization 2 and from the correctness of the algorithm DCL-MLSM-CliqueTree.

5.3. MLS on a Non-Chordal Graph

A MLS ordering of a non-chordal graph G is not necessarily an MEO of G. It was shown in [8] that LexBFS ends on a moplex, i.e., if α is a LexBFS ordering of a non-clique graph G, then there is a moplex X_1 of G such that $X_1 = \{\alpha(1), \ldots, \alpha(|X_1|)\}$. As the restriction of α to $V \setminus X_1$ is a LexBFS ordering of $G(V \setminus X_1)$, it follows that α is compatible with a simple moplex partition of G. However, if G is not chordal, then α is not necessarily a PMO of G, and it is not necessarily an MMO of G and not even an MEO of G.

Example 4. *Let G be the graph shown in Figure 4, and let $\alpha = (1,2,3,4,5)$. α is a LexBFS ordering of G (the final labels are indicated). α is compatible with the simple moplex partition $(\{1\},\{2,3\},\{4,5\})$. However, α is not an MEO of G since the graph $H = G^+_\alpha$ (shown in the figure with dashed fill edges) is not a minimal triangulation of G.*

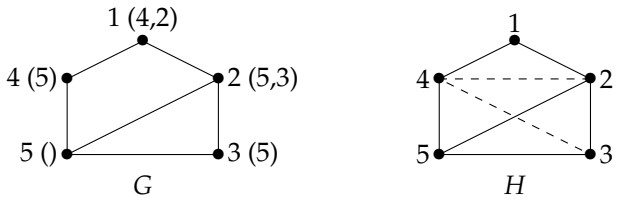

Figure 4. Lexicographic Breadth-First Search (LexBFS) on a non-chordal graph computes an ordering that is compatible with a simple moplex partition, but not with an MEO.

The work from [8] also showed that if α is a LexBFS ordering of a graph G, then the minimal separators included in $N(\alpha(1))$ are totally ordered by inclusion and that α consecutively numbers the vertices of each connected component of $G(V \setminus N[\alpha(1)])$ and its neighborhood. More accurately, there is an order $(C_1, \ldots, C_p)$ on the connected component of $G(V \setminus N[\alpha(1)])$, such that for each i in $[1, p-1]$; $N(C_i) \subseteq N(C_{i+1})$ (the minimal separators included in $N(\alpha(1))$ are the sets $N(C_i)$), and α is compatible with the ordered partition $(X_1, N[C_p] \setminus N(C_{p-1}), \ldots, N[C_2] \setminus N(C_1), N[C_1])$, where X_1 is the moplex containing $\alpha(1)$. Xu et al. [14] showed that LexDFS also ends on a moplex of the input graph G (and therefore, a LexDFS ordering is compatible with a simple moplex partition of G), but they left open the question whether LexDFS orderings also have the properties on minimal separators and connected components. The following counterexample shows that these properties of LexBFS do not extend to LexDFS.

Counterexample 2. *Let G be the graph shown in Figure 5, and let $\alpha = (1,2,\ldots,,7)$. α is a LexDFS ordering of G (the final labels are indicated). The minimal separators included in $N[1]$ are $\{2,6\}$ and $\{4,6\}$ and are incomparable for inclusion. If the property on connected components was true, C_1 would be the component numbered last by α, i.e., $\{7\}$, but the vertices of $N[\{7\}]$ (7, 6 and 2) are not numbered consecutively by α. If G is the graph shown in Figure 6 with $\alpha = (1,2,\ldots,,6)$, even the restriction of α to $V \setminus N[1]$ is not compatible with a sequence of the connected components of $G(V \setminus N[1])$.*

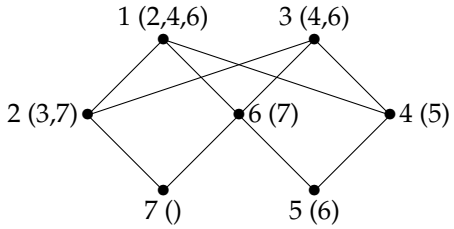

Figure 5. With LexDFS, the minimal separators included in $N(\alpha(1))$ are not totally ordered by inclusion.

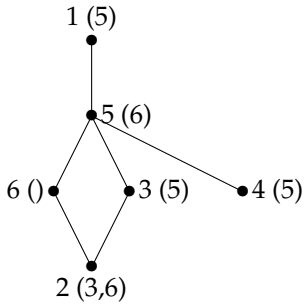

Figure 6. LexDFS does not number the connected components of $G(V \setminus N[\alpha(1)])$ one after another.

It is shown in [12] that for each labeling structure $\mathcal{L}$ with a total order on labels and each $\mathcal{L}$-MLS ordering α of a graph G, $\alpha(1)$ is an OCF-vertex of G, i.e., satisfies the property: for each pair $\{y,z\}$ of non-adjacent vertices in $N(\alpha(1))$, there is a connected component C of $G(V \setminus N[\alpha(1)])$ such that $y,z \in N_G(C)$. However, the definition of a labeling structure given in [12] is less general than the definition given in this paper, as IC is replaced by conditions (p1): $l \prec Inc(l,i)$ and (p2): $if\ l \prec l'\ then\ Inc(l,i)l \prec Inc(l',i)$: (p1) and (p2) imply IC and IC implies (p1), but does not imply (p2). It turns out that the property of $\alpha(1)$ being an OCF-vertex does not extend to a labeling structure defined with IC instead of (p1) and (p2) (a counterexample can be built with some effort). It is the only result from [12] that does not extend to a labeling structure defined with IC.

6. Conclusions

In this paper, we explain how a clique tree of a chordal graph H can be computed from an arbitrary PEO of H, from a PMO of H and from a modification of the algorithm MLS. We show that a PMO allows one to build the cliques of a clique tree one after another. We characterize labeling structures for which it is possible to detect the beginning of a new clique using the labels and show that each labeling structure can detect this with labels when building a clique tree of the complement graph.

Some results concerning algorithm MLS in Section 3 and MLSM in Section 5 are generalizations of the results already known for MCS, LexBFS or MCS-M. The proofs in this paper largely use Inclusion Condition (IC) (through Lemma 1) and make it clear that IC is fundamental for the properties of the computed orderings. We believe that many results proven for some particular instances of MLS or MLSM can be proven in a more general way for MLS or MLSM using IC. This has already been done in [10,12,20]. We leave open the question of which other results could be generalized using IC and which applications could follow from these generalizations.

Acknowledgments: We thank the reviewers for their helpfull comments and suggestions.

Author Contributions: Anne Berry and Geneviève Simonet both provided the contents and wrote the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dirac, G.A. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburg* **1961**, *25*, 71–76.
2. Fulkerson, D.R.; Gross, O.A. Incidence matrixes and interval graphs. *Pac. J. Math.* **1965**, *15*, 835–855.
3. Gavril, F. The Intersection Graphs of Subtrees in Trees Are Exactly the Chordal Graphs. *Combin. Theory Ser. B* **1974**, *16*, 47–56.
4. Blair, J.R.S.; Peyton, B.W. An introduction to chordal graphs and clique trees. *Graph Theory Sparse Matrix Comput.* **1993**, *84*, 1–29.
5. Tarjan, R.E.; Yannakakis, M. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.* **1984**, *13*, 566–579.
6. Rose, D.J.; Tarjan, R.E.; Lueker, G.S. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.* **1976**, *5*, 266–283.
7. Kumar, P.S.; Madhavan, C.E.V. Minimal vertex separators of chordal graphs. *Discret. Appl. Math.* **1998**, *89*, 155–168.
8. Berry, A.; Bordat, J.P. Separability generalizes Dirac's theorem. *Discret. Appl. Math.* **1998**, *84*, 43–53.
9. Corneil, D.G.; Krueger, R. A unified view of graph searching. *SIAM J. Discret. Math.* **2008**, *22*, 1259–1276.
10. Berry, A.; Krueger, R.; Simonet, G. Maximal Label Search Algorithms to Compute Perfect and Minimal Elimination Orderings. *SIAM J. Discret. Math.* **2009**, *23*, 428–446.
11. Krueger, R.; Simonet, G.; Berry, A. A General Label Search to investigate classical graph search algorithms. *Discret. Appl. Math.* **2011**, *159*, 128–142.
12. Berry, A.; Blair, J.R.S.; Bordat, J.-P.; Simonet, G. Graph extremities defined by search algorithms. *Algorithms* **2010**, *3*, 100–124.
13. Berry, A.; Pogorelcnik, R. A simple algorithm to generate the minimal separators and the maximal cliques of a chordal graph. *Inform. Process. Lett.* **2011**, *111*, 508–511.
14. Xu, S.-J.; Li, X.; Liang, R. Moplex orderings generated by the LexDFS algorithm. *Discret. Appl. Math.* **2013**, *161*, 2189–2195.
15. Berry, A.; Pogorelcnik, R.; Simonet, G. An introduction to clique minimal separator decomposition. *Algorithms* **2010**, *3*, 197–215.
16. Berry, A.; Wagler, A. Triangulation and clique separator decomposition of claw-free graphs. In Proceedings of the WG 2012, Jerusalem, Israel, 26–28 June 2012; pp. 7–21.
17. Brandstädt, A.; Hoàng, C.T. On clique separators, nearly chordal graphs, and the Maximum Weight Stable Set Problem. *Theoret. Comput. Sci.* **2007**, *389*, 295–306.
18. Leimer, H.-G. Optimal decomposition by clique separators. *Discret. Math.* **1993**, *113*, 99–123.
19. Tarjan, R.E. Decomposition by clique separators. *Discret. Math.* **1985**, *55*, 221–232.
20. Berry, A.; Pogorelcnik, R.; Simonet, G. Organizing the atoms of the clique separator decomposition into an atom tree. *Discret. Appl. Math. Math.* **2014**, *177*, 1–13.
21. Berry, A.; Blair, J.R.S.; Heggernes, P.; Peyton, B.W. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica* **2004**, *39*, 287–298.
22. Brandstädt, A.; Le, V.B.; Spinrad, J.P. *Graph Classes: A Survey*; Monographs on Discrete Mathematics and Applications; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 1999; Volume 3.
23. Golumbic, M.C. *Algorithmic Graph Theory and Perfect Graphs*; Academic Press: New York, NY, USA, 1980.
24. Berry, A.; Bordat, J.P. Moplex elimination orderings. *Electron. Notes Discret. Math.* **2001**, *8*, 6–9.
25. Rose, D.J. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.* **1970**, *32*, 597–609.
26. Spinrad, J.P. *Efficient Graph Representations*; Fields Institute Monographs, American Mathematical Society: Presidence, RI, USA, 2003.