Maximal Label Search Algorithms to Compute Perfect and Minimal Elimination Orderings

Anne Berry, Richard Krueger, Geneviève Simonet

Maximal Label Search algorithms to compute perfect and minimal elimination orderings

A. Berry * R. Krueger † G. Simonet ‡

September 2, 2008

Abstract

Many graph search algorithms use a vertex labeling to compute an ordering of the vertices. We examine such algorithms which compute a peo (perfect elimination ordering) of a chordal graph, and corresponding algorithms which compute an meo (minimal elimination ordering) of a non-chordal graph, an ordering used to compute a minimal triangulation of the input graph.

We express all known peo-computing search algorithms as instances of a generic algorithm called MLS (Maximal Label Search) and generalize Algorithm MLS into CompMLS, which can compute any peo.

We then extend these algorithms to versions which compute an meo, and likewise generalize all known meo-computing search algorithms. We show that not all minimal triangulations can be computed by such a graph search, and, more surprisingly, that all these search algorithms compute the same set of minimal triangulations, even though the computed meos are different.

Finally, we present a complexity analysis of these algorithms. [1]

Keywords: Graph search, peo, meo, minimal triangulation, elimination scheme, MLS.

1 Introduction

Graph searching plays a fundamental role in many algorithms, particularly using Breadth-First or Depth-First searches and their many variants. One important application is to compute special graph orderings related to the chordality of a graph. When the input graph is chordal, one wants to find an ordering of the vertices called a *peo* (perfect elimination ordering), which repeatedly selects a vertex whose neighborhood is a clique (called a *simplicial vertex*), and removes it from the graph. This is a certificate of chordality, as,

*LIMOS, Ensemble scientifique des Cézeaux, F-63177 Aubière, France, fax 00 33 4 73 40 76 39. berry@isima.fr

†Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 3G4. krueger@cs.toronto.edu

‡LIRMM, 161, Rue Ada, F-34392 Montpellier, France. simonet@lirmm.fr

[1] An extended abstract of part of this paper was published in WG 2005 [4]

given an ordering of the vertices, one can determine in linear time whether it is a peo of the graph.

When the input graph fails to be chordal, it is often interesting to embed it into a chordal graph by adding an inclusion-minimal set of edges, a process called *minimal triangulation*. One of the ways of accomplishing this is to use an ordering of the vertices called an *meo* (minimal elimination ordering), and use this to simulate a peo by repeatedly adding any edges whose absence would violate the simplicial condition.

Though some earlier work had been done on these problems ([14, 13]), the seminal paper is that of Rose, Tarjan and Lueker [15], which presented two very efficient algorithms to compute a peo or an meo. They introduced the concept of *lexicographic order* (which roughly speaking is a dictionary order), and used this for graph searches which at each step choose an unnumbered vertex of maximal label. With this technique, they introduced Algorithm LEX M, which for a non-chordal graph $G = (V, E)$ computes an meo in a very efficient $O(nm)$ time, (where $n = |V|$ and $m = |E|$), and then streamlined this for use on a chordal graph, introducing what is now called Algorithm LexBFS, a Breadth-First Search which runs in optimal $O(n+m)$ time and computes a peo if the input graph is chordal (see a survey on LexBFS in [8]).

Later work has been done on computing peos. Tarjan and Yannakakis [17] presented Algorithm MCS (Maximal Cardinality Search) which is similar to LexBFS but uses a simplified labeling (a cardinality choice criterion is used instead of a lexicographic one). M CS also computes in linear time a peo if the input graph is chordal.

Shier [16] remarks that neither LexBFS nor MCS is capable of computing all peos. He proposes Algorithms MEC and MCC, which can both compute any peo of a chordal graph.

Recently, Corneil and Krueger [9] introduced Algorithm LexDFS as a Depth-First analogue to LexBFS. They also introduced Algorithm MNS (Maximal Neighborhood Search) which chooses at each step a vertex whose set of numbered neighbors is inclusion-maximal. They gave characterizations of the orderings computed by these search algorithms and observed, from a result of Tarjan and Yannakakis [17] on the property characterizing MNS orderings, that every MNS ordering yields a peo if the input graph is chordal. They showed with these characterizations that any ordering computed by LexBFS, MCS or LexDFS can also be computed by MNS.

Berry, Blair, Heggernes and Peyton [1] recently introduced Algorithm MCS-M, which computes an meo. MCS-M is extended from MCS in the same fashion LEX M can be extended from LexBFS. The sets of meos defined by LEX M and by MCS-M are different, but Villanger [20] recently showed that the same sets of minimal triangulations were obtained.

In this paper, we address natural questions which arise about peos and meos: how can the existing algorithms be generalized? Do these new algorithms compute all peos of a chordal graph? Can they all be extended to compute meos? What sets of minimal triangulations are obtained?

Algorithms LexBFS, MCS, LexDFS and MNS clearly process in a similar

way: they number the vertices of the input graph by repeatedly numbering an unnumbered vertex with maximal label and incrementing the labels of its neighbors. They only differ by their vertex labeling, i.e. the nature of labels and the way they are compared, initialized and incremented. We show that they can be described as instances of a generic algorithm called MLS (Maximal Label Search) having the vertex labeling as a parameter. We show that every instance of MLS computes a peo of a chordal graph, but cannot compute every peo of every chordal graph. In order to obtain all possible peos, we extend MLS to CompMLS, which uses Shier's idea of working on the connected components of the subgraph induced by the unnumbered vertices. We show that every instance of generic CompMLS is capable of computing any peo of a chordal graph.

We then go on to examine the issues pertaining to meos and minimal triangulations. We show that MNS, MLS and CompMLS can all be extended to compute an meo, in the same way that LEX M is extended from LexBFS. We show the very strong result that all the sets of minimal triangulations computed are the same, independent of the meo-computing algorithm which is used, and that not all minimal triangulations can be computed by this new family of algorithms.

The paper is organized as follows: Section 2 gives some definitions and notations, in Section 3 we discuss peos, in Section 4 we discuss meos and in Section 5 we present a complexity analysis of the algorithms defined in the paper.

2 Preliminaries

All graphs in this work are undirected and finite. A graph is denoted $G = (V, E)$, with $n = |V|$, and $m = |E|$. The *neighborhood* of a vertex x in G is denoted $N_G(x)$, or simply $N(x)$ if the meaning is clear. An ordering on V is a one-to-one mapping from $\{1, 2, ..., n\}$ to V. In every figure in this paper showing an ordering α on V, α is defined by giving on the figure the number $\alpha^{-1}(x)$ for every vertex x. $\mathbb{Z}^+$ denotes the set of positive integers $\{1, 2, 3, \dots\}$ and for any positive integer i, $\mathbb{Z}_i^+$ denotes the set of positive integers strictly larger than i.

A *chordal* (or *triangulated*) graph is a graph with no chordless cycle of length greater or equal to 4. To recognize chordal graphs efficiently, Fulkerson and Gross [12] used a greedy elimination structure on simplicial vertices: "A graph is chordal iff one can repeatedly find a simplicial vertex and delete it from the graph, until no vertex is left" (a vertex is simplicial if its neighborhood is a clique). This defines an ordering on the vertices which is called a *perfect elimination ordering (peo)* of the graph.

When a graph G fails to be chordal, any ordering α on the vertices can be used to embed G into a chordal graph (called a *triangulation* of G) by repeatedly choosing the next vertex x, adding any edges necessary to make it simplicial, and removing x. If F is the set of added edges, the graph obtained is chordal and is denoted $H = (V, E + F) = G_\alpha^+$.

If $H = (V, E + F)$ is a triangulation of $G = (V, E)$, and if for every proper subset $F' \subset F$, graph $(V, E + F')$ fails to be chordal, H is called a *minimal triangulation* of G. If moreover α is an ordering such that $H = G^+_\alpha$, α is called a *minimal elimination ordering (meo)* of G.

In [15], two very important characterizations are given:

Path Lemma
For any graph $G = (V, E)$, any ordering α on V and any x, y in V such that $\alpha^{-1}(y) < \alpha^{-1}(x)$, xy is an edge of G^+_α iff there is a path μ in G from x to y such that $\forall t \in \mu \setminus \{x, y\},\ \alpha^{-1}(t) < \alpha^{-1}(y)$.

Unique Chord Property
For any graph $G = (V, E)$ and any triangulation $H = (V, E + F)$ of G, H is a minimal triangulation of G iff each edge in F is the unique chord of a 4-cycle of H.

3 Computing peos

Everyone of Algorithms LexBFS, MCS, LexDFS and MNS works in the following fashion: start with a graph where all vertices are unnumbered and have the same label. Repeatedly choose an unnumbered vertex x whose label is maximal (with respect to a given partial order on labels), give x the following number i (in increasing or decreasing order according to the algorithm), and increment the label of each as yet unnumbered neighbor of x into a new value depending.on its current value and i. Algorithms LexBFS and MCS, as defined in [15] and [17], number vertices from n down to 1 whereas LexDFS and MNS, as defined in [9], number vertices from 1 to n, so that they actually compute the reverse of a peo of every chordal graph. In this paper, our algorithms compute peos and meos directly, and thus number vertices from n down to 1: thus vertex number 1 of a peo-computing algorithm will be a simplicial vertex of the graph.

In order to define a generic peo-computing algorithm, we first define a *labeling structure*:

Definition 3.1 *A **labeling structure** is a structure $(L, \preceq, l_0, Inc)$, 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), which will be used to choose a vertex of maximal label,*
- *l_0 is an element of L, which will be used to initialize the labels,*
- *Inc is a mapping from $L \times \mathbb{Z}^+$ to L, which will be used to increment a label, and such that the following condition IC (Inclusion Condition) holds:*

IC : for any subsets I and I' of $\mathbb{Z}_2^+$, if $I \subset I'$ then $lab_S(I) \prec lab_S(I')$,
where $lab_S(I) = Inc(...(Inc(Inc(l_0, i_1), i_2), ...), i_k)$,
where $I = \{i_1, i_2, ..., i_k\}$ with $i_1 > i_2 > ... > i_k$.

It will sometimes be useful to use the number n of vertices of the graph to be labeled in the definition of $Inc(l, i)$. It will be the case for instance for the labeling structure S_3 associated with Algorithm LexDFS. In that case, Inc can be seen as a family of mappings Inc_n from $L \times \mathbb{Z}^+$ to L for each positive integer n.

The corresponding algorithm, which we introduce as MLS (Maximal Label Search), is given by Figure 1. MLS iteratively selects a vertex to add to the ordering and increments the labels of its unselected neighbors. We will refer to the iteration of the loop that defines $\alpha(i)$ as Step i of the algorithm.

Algorithm MLS (Maximal Label Search)
input : A graph $G = (V, E)$ and a labeling structure $(L, \preceq, l_0, Inc)$.
output: An ordering α on V.
Initialize all labels as l_0; $G' \leftarrow G$;
for $i = n$ **downto** 1 **do**
  Choose a vertex x of G' of maximal label;
  $\alpha(i) \leftarrow x$;
  foreach y *in* $N_{G'}(x)$ **do**
    $label(y) \leftarrow Inc(label(y), i)$;
  Remove x from G';

Figure 1: Algorithm MLS.

LexBFS, MCS, LexDFS and MNS are all special cases of MLS, with the following labeling structures $(L, \preceq, l_0, Inc)$; in each case, we also give the value of $lab_S(I)$ for any subset I of $\mathbb{Z}^+$.

LexBFS (Structure S_1): L is the set of lists of elements of $\mathbb{Z}^+$, $\preceq$ is the lexicographic order (a total order), l_0 is the empty list, $Inc(l, i)$ is obtained from l by adding i to the end of the list, $lab_{S_1}(I)$ is the string of the integers in I in decreasing order.

MCS (Structure S_2): $L = \mathbb{Z}^+ \cup \{0\}$, $\preceq$ is $\leq$ (a total order), $l_0 = 0$, $Inc(l, i) = l + 1$, $lab_{S_2}(I) = |I|$.

LexDFS (Structure S_3): L is the set of lists of elements of $\mathbb{Z}^+$, $\preceq$ is the lexicographic order (a total order), l_0 is the empty list, $Inc(l, i)$ is obtained from l by adding $n + 1 - i$ to the beginning of the list, $lab_{S_3}(I)$ is the string of the complements to $n + 1$ of the integers in I in decreasing order.

MNS (Structure S_4): L is the power set of $\mathbb{Z}^+$, $\preceq$ is $\subseteq$ (not a total order), $l_0 = \emptyset$, $Inc(l, i) = l \cup \{i\}$, $lab_{S_4}(I) = I$.

In our proofs, we will use the following notations.

Notations 3.2 *For any graph $G = (V, E)$, any execution of our algorithms on G computing some ordering α on V, and any integer i between 1 and n,*

- V_i is the set of still unnumbered vertices at the beginning of Step i, i.e. the set $\{\alpha(j),\ 1 \leq j \leq i\}$,

- G'_i is graph G' at the beginning of Step i, i.e. the subgraph of G induced by V_i,

and, for each $y \in V_i$,

- $label_i(y)$ is the value of $label(y)$ at the beginning of Step i and

- $Num_i(y) = \{j \in \{i+1, i+2, ..., n\} \mid label(y)$ has been incremented at Step $j\}$.

The following Lemma is clear from Algorithm MLS:

Lemma 3.3 *For any graph $G = (V, E)$, any labeling structure S, any execution of MLS on G and S computing some ordering α on V, any integer i between 1 and n and any $y \in V_i$, $label_i(y) = lab_S(Num_i(y))$, and $Num_i(y) = Num^{\alpha}_{G,i}(y)$, where $Num^{\alpha}_{G,i}(y)$ denotes the set of integers $j > i$ such that $\alpha(j)$ is adjacent to y in G.*

Thus the label of y at the beginning of Step i is equal to $lab_S(Num^{\alpha}_{G,i}(y))$, where $Num^{\alpha}_{G,i}(y)$ is defined from the ordering α computed so far on numbered vertices, independently from the labeling structure involved. This property will allow us to characterize the orderings computed by MLS, and to compare the sets of orderings computed with different labeling structures (Characterization 3.4 and Lemma 3.5).

We can view MLS as a generic algorithm with parameter S. For every labeling structure S, we denote by S-MLS the instance of generic Algorithm MLS using this particular labeling structure S and by "S-MLS ordering of a graph G" any ordering that can be computed by S-MLS on input graph G. Thus, LexBFS is S_1-MLS, MCS is S_2-MLS, LexDFS is S_3-MLS, and MNS is S_4-MLS.

The set of S-MLS orderings of a given graph depends on S. An MLS ordering of a graph G is an ordering that can be computed by MLS on G, i.e. by S-MLS for some labeling structure S. Thus, the set of MLS orderings of G is the union of the sets of S-MLS orderings of G for all labeling structures S.

The following theorem shows that MNS can compute every S-MLS ordering of a given graph for every labeling structure S. This theorem can be proved using the MNS characterization presented in [9]. We will prove it by using the following more general results.

Characterization 3.4 *For any graph G, any labeling structure S, and any ordering α of V, α is an S-MLS ordering of G if and only if for any integers i, j such that $1 \leq j < i \leq n$, $lab_S(Num^{\alpha}_{G,i}(\alpha(i))) \not< lab_S(Num^{\alpha}_{G,i}(\alpha(j)))$.*

Proof: α is an S-MLS ordering of G if and only if for any integer i between 1 and n, the label of $\alpha(i)$ at the beginning of Step i is maximal among the labels of vertices $\alpha(j)$, $i \leq j \leq n$. We conclude with Lemma 3.3. □

Lemma 3.5 *Let S and S' be labeling structures with partial orders $\preceq_S$ and $\preceq_{S'}$ resp. such that for any subsets I and I' of $\mathbb{Z}_2^+$, if $lab_{S'}(I) \prec_{S'} lab_{S'}(I')$ then $lab_S(I) \prec_S lab_S(I')$.*
Then every S-MLS ordering of G is also an S'-MLS ordering of G for every graph G.

Proof: Let G be a graph and α be an S-MLS ordering of G. By Characterization 3.4, for any integers i, j such that:
$1 \leq i < j \leq n$, $lab_S(Num^{\alpha}_{G,i}(\alpha(i))) \not\prec_S lab_S(Num^{\alpha}_{G,i}(\alpha(j)))$,
where $Num^{\alpha}_{G,i}(\alpha(i))$ and $Num^{\alpha}_{G,i}(\alpha(j))$ are subsets of $\mathbb{Z}_2^+$ since $i > 1$, so $lab_{S'}(Num^{\alpha}_{G,i}(\alpha(i))) \not\prec_{S'} lab_{S'}(Num^{\alpha}_{G,i}(\alpha(j)))$. By Characterization 3.4 again α is an S'-MLS ordering of G. □

Theorem 3.6 *For any graph $G = (V, E)$ and any labeling structure S, any S-MLS ordering of G is an MNS ordering of G.*

Proof: This follows immediately from Lemma 3.5 and condition IC, since MNS = S_4-MLS with $lab_{S_4}(I) = I$ and $\prec_{S_4} = \subset$. □

A corollary of Theorem 3.6 is that any instance of MLS computes a peo of a chordal graph, since this is true for MNS [9].

Another consequence is that any LexBFS, MCS or LexDFS ordering of a graph is also an MNS ordering, which already follows from the characterizations given in [9]. However, for arbitrary labeling structures S and S', an ordering computed by S-MLS need not be computable by S'-MLS. For instance, Figure 2(a) shows a LexBFS ordering which is not an MCS ordering, while Figure 2(b) shows an MCS ordering which is not a LexBFS ordering. There also exist graphs with MNS orderings that are neither LexBFS nor MCS orderings.

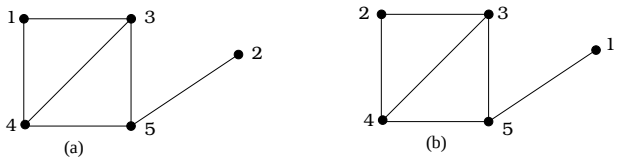

Figure 2: A chordal graph with different (a) LexBFS and (b) MCS orderings.

As the set of MLS orderings of a graph G is the union of the sets of S-MLS orderings of G for all labeling structures S and as MNS is equal to S_4-MLS, it follows from Theorem 3.6 that every graph has the same MLS and MNS orderings. However, be careful that MLS and MNS are different algorithms since MLS has a graph and a labelling structure as input whereas MNS only has a graph (or, if MLS is seen as a generic algorithm with a labeling structure as a parameter, MNS is only an instance of MLS). Figure 3 gives two different relations on peo-computing algorithms. Figure 3 (a) shows

some instances of generic Algorithm MLS, each arrow from MLS to one of its instances being labeled with the corresponding value of parameter S. Figure 3 (b) shows the inclusion order on the sets of orderings computable by these algorithms on a given graph. In this figure, S is an arbitrary labeling structure.

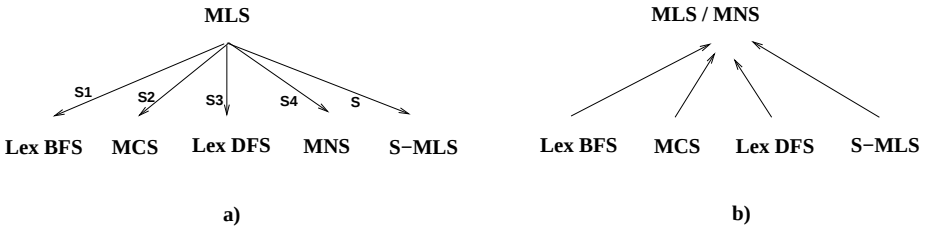

Figure 3: (a) Instances of MLS and (b) Inclusion order on the sets of computable orderings.

It is interesting to remark that even though MLS, or equivalently MNS, is more general (in the sense that it can compute more peos) than LexBFS and MCS, it still is not powerful enough to compute every possible peo of a given chordal graph. This is shown by the simple counterexample in Figure 4: no MLS execution on this graph will find the ordering indicated, although it is clearly a peo.

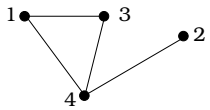

Figure 4: α is a CompMNS ordering of G but not an MNS one.

In order to make it possible to find *any* peo, we further generalize MLS using Shier's idea [16] of using the *connected components* of the subgraph G' induced by the unnumbered vertices. We thus introduce Algorithm CompMLS, defined from Algorithm MLS by replacing:
"Choose a vertex x of G' of maximal label;"
with
"Choose a connected component C of G';
Choose a vertex x of C of maximal label in C;".

This generalizes the entire family of peo-computing algorithms discussed in this paper: for any X in {LexBFS, MCS, LexDFS, MNS, MLS}, Algorithm CompX is a generalization of X, and we will show that it computes a peo if the graph is chordal. Algorithms MEC and MCC defined by Shier [16] are instances of generic Algorithm CompMLS: MEC is CompMNS, i.e. S_4- CompMLS, and MCC is CompMCS, i.e. S_2- CompMLS. Algorithm CompMNS can compute the peo of Figure 4. In fact, Shier proved in [16] that CompMNS and even CompMCS compute all peos of a chordal graph. We show that this holds for every instance of Algorithm CompMLS, using

some results from Section 4.

Theorem 3.7 *For any chordal graph G and any labeling structure S, the S-CompMLS orderings of G are exactly its peos.*

Proof: Let G be a chordal graph and S be a labeling structure. By [16] the CompMNS orderings of G are exactly its peos, and by Theorem 4.15 from Section 4, G has the same S-CompMLSM and CompMNSM orderings, which are also its S-CompMLS and CompMNS orderings since G is chordal (by the extension of Property 4.7 from Section 4 to CompMLS and CompMLSM). □

We consider here that a labeling structure is defined without the condition IC, and we discuss the choice of the condition IC in view of obtaining an algorithm computing peos of chordal graphs. By Theorem 3.6 IC is a sufficient condition on a labeling structure S for S-MLS to compute only peos of every chordal graph. It turns out that it is also a necessary one, so that IC is exactly the condition required on a labeling structure for MLS to compute only peos of every chordal graph.

Theorem 3.8 *The condition IC imposed on a labeling structure S is a necessary and sufficient condition for S-MLS to compute only peos of every chordal graph.*

Proof: IC is a sufficient condition since by Theorem 3.6, S-MLS only computes MNS orderings, and therefore peos, of every chordal graph.
Conversely, suppose there are some subsets I and I' of $\mathbb{Z}_2^+$ such that $I \subset I'$ and $lab_S(I) \not\prec lab_S(I')$, and let us show that there is a chordal graph G and an S-MLS ordering of G that is not a peo of G. Let $q = max(I')$. We choose two subset I and I' of $\mathbb{Z}_2^+$ such that $I \subset I'$, $lab_S(I) \not\prec lab_S(I')$, $max(I') = q$ and $min(I')$ is the largest possible with these conditions. Let $p = min(I')$. $p > 2$ since I' is a subset of of $\mathbb{Z}_2^+$. Let $G = (V, E)$ with $V = \{z_1, z_2, ..., z_q\}$ and $E = \{z_i z_j, p \leq i < j \leq q\} \cup \{z_i z_{p-1}, i \in I'\} \cup \{z_i z_{p-2}, i \in I\} \cup \{z_{p-1} z_{p-2}\}$. G is chordal since $(z_1, z_2, ..., z_q)$ is a peo of G. By the choice of I and I' there is an execution of S-MLS on G choosing z_q, z_{q-1}, ... z_p first, and then choosing z_{p-2} before z_{p-1}. The resulting S-MLS ordering of G is not a peo of G because the set of neighbors of z_{p-1} with higher numbers than z_{p-1} in this ordering is not a clique, since z_{p-2} is not adjacent to the vertices of the form z_i, $i \in I' \setminus I$. □

The condition imposed on a labeling structure was defined in a different way in [4]. Instead of satisfying IC, the mapping Inc had to satisfy the following condition: for any integer n in $\mathbb{Z}^+$, any integer i between 1 and n and any labels l and l' in L_i^n, the following properties hold:

(ls1) $l \prec Inc(l, i)$

(ls2) if $l \prec l'$ then $Inc(l, i) \prec Inc(l', i)$

where L_i^n is the subset of L defined by induction on i by:

$L_n^n = \{l_0\}$, and

$L^n_{i-1} = L^n_i \cup \{l = Inc(l', i) \mid l' \in L^n_i\}$, for any i from n down to 2.

It is easy to show that this condition implies IC, but the converse is not true. For instance, let S be the labeling structure obtained from S_4 (the structure used for MNS) by replacing the inclusion order $\preceq_{S_4}$ by the partial order $\preceq$ on L defined by: for any $l, l' \in L$, $l \preceq l'$ iff ($l \subseteq l'$ or ($l = \{4\}$ and $5 \in l'$)) (checking that $\preceq$ is a partial order on L is left to the reader). IC holds since the inclusion order is a suborder of $\preceq$, but not (ls2) since $\{4\} \prec \{5\}$ but $Inc(\{4\}, 3) = \{4, 3\} \not\prec \{5, 3\} = Inc(\{5\}, 3)$. Thus IC is more appropriate than the conjunction of (ls1) and (ls2) in the context of peo-computing algorithms, though a labeling structure satisfying IC necessarily satisfies (ls1) and often satisfies (ls2) in practice.

Let us conclude this section by some remarks on running the MLS family of algorithms on *non-chordal* graphs. LexBFS has been used on AT-free graphs [10] and has been shown to have very interesting invariants even on an arbitrary graph ([2, 3]). Likewise, MCS has also been used on various graph classes ([11, 6, 7]).

Unlike a chordal graph, a non-chordal graph does not necessarily have the same CompLexBFS, CompMCS, CompLexDFS, and CompMNS orderings. Figure 5(a) shows a CompLexBFS ordering which is not a CompMCS one, while Figure 5(b) shows a CompMCS ordering which is not a CompLexBFS one.

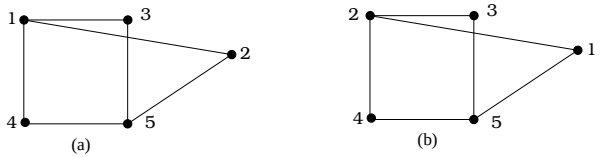

Figure 5: A non-chordal graph with different (a) CompLexBFS and (b) CompMCS orderings.

4 Computing meos

We will now introduce the extensions of Algorithms MNS, MLS and CompMLS into their meo-computing counterparts.

To extend LexBFS into LEX M, at each step choosing a vertex x of maximum label $label(x)$, an edge is added between x and any unnumbered vertex y whenever there is a path from x to y in the subgraph induced by the unnumbered vertices such that all internal vertices on the path have a label strictly smaller than the label of y. This approach has been used recently in [1] to extend MCS into meo-computing Algorithm MCS-M; here, we extend MLS into MLSM, as given by Figure 6. Thus LEX M is S_1-MLSM, MCS-M is S_2-MLSM, LexDFS-M is defined as S_3-MLSM, and MNSM is defined as S_4-MLSM. We will see that Algorithm MNSM is in fact as general as MLSM: every MLSM ordering of a graph is an MNSM ordering.

Algorithm MLSM (Maximal Label Search for Meo)

input : A graph $G = (V, E)$ and a labeling structure $(L, \preceq, l_0, Inc)$.
output: An meo α on V and a minimal triangulation $H = G^+_\alpha$ of G.

Initialize all labels as l_0; $E' \leftarrow \emptyset$; $G' \leftarrow G$;
for $i = n$ **downto** 1 **do**
  Choose a vertex x of G' of maximal label;
  $\alpha(i) \leftarrow x$;
  foreach *vertex* y *of* G' *different from* x **do**
    if *there is a path from* x *to* y *in* G' *such that every internal vertex on the path has a label strictly smaller than* $label(y)$ **then**
      $E' \leftarrow E' \cup \{xy\}$;
  foreach y *in* V *such that* $xy \in E'$ **do**
    $label(y) \leftarrow Inc(label(y), i)$;
  Remove x from G';
$H \leftarrow (V, E')$;

Figure 6: Algorithm MLSM.

For any labeling structure S, we call S-MLSM the instance of Algorithm MLSM using S, and S-MLSM ordering an ordering computed by S-MLSM.

Clearly, the relation between $label_i(y)$ and $Num_i(y)$ in an execution of MLS still holds in an execution of MLSM.

Lemma 4.1 *For any graph $G = (V, E)$, any labeling structure S, any execution of MLSM on G and S, any integer i between 1 and n and any $y \in V_i$, $label_i(y) = lab_S(Num_i(y))$.*

We will show that as for MLS, $Num_i(y)$ can be defined from the ordering α computed so far on numbered vertices, independently from the labeling structure involved, with similar consequences in terms of characterizing MLSM orderings and comparing the sets of orderings computed by MLSM with different labeling structures.

4.1 The MLSM family of algorithms

Theorem 4.2 *For any execution of MLSM, $H = G^+_\alpha$, and α is a meo of G.*

To prove this, we will need several technical lemmas. Lemma 4.3 is clear from algorithm MLSM, Lemma 4.4 immediately follows from Lemma 4.1 and condition IC. The proof of Lemma 4.5 is long and technical, and so for reasons of readability is given in the Appendix.

Lemma 4.3 *For any graph $G = (V, E)$, any execution of MLSM on G computing ordering α and graph H, any integers i, j such that $1 \leq i < j \leq n$ and any y in V_i, the following propositions are equivalent:*

1. $j \in Num_i(y)$,
2. $\alpha(j)y$ *is an edge of* H,
3. *there is a path* μ *in* G'_j *from* $\alpha(j)$ *to* y *such that* $\forall t \in \mu \setminus \{\alpha(j), y\}$, $label_j(t) \prec label_j(y)$.

Lemma 4.4 *For any graph* $G = (V, E)$, *any execution of MLS or MLSM on* G, *any integer* i *between* 1 *and* n *and any* x, y *in* V_i,
(i) if $Num_i(x) = Num_i(y)$ *then* $label_i(x) = label_i(y)$, *and*
(ii) if $Num_i(x) \subset Num_i(y)$ *then* $label_i(x) \prec label_i(y)$.

Lemma 4.5 *For any graph* G, *any execution of MLSM on* G *computing ordering* α, *any integer* i *between* 1 *and* n *and any path* μ *in* G'_i *ending in some vertex* y,

a) $\forall t \in \mu \backslash \{y\}$, $label_i(t) \prec label_i(y)$ *iff* $\forall t \in \mu \backslash \{y\}$, $Num_i(t) \subset Num_i(y)$,

b) if $\forall t \in \mu \backslash \{y\}$, $label_i(t) \prec label_i(y)$ *then* $\forall t \in \mu \backslash \{y\}$, $\alpha^{-1}(t) < \alpha^{-1}(y)$,

c) if $\forall t \in \mu \setminus \{y\}$, $\alpha^{-1}(t) < \alpha^{-1}(y)$ *then* $\forall t \in \mu \setminus \{y\}$, $Num_i(t) \subseteq Num_i(y)$.

Proof: [of Theorem 4.2] We first show that for any execution of MLSM, $H = G^+_\alpha$. Let $x, y \in V$ such that $\alpha^{-1}(y) < \alpha^{-1}(x) = i$. Let us show that xy is an edge of H iff it is an edge of G^+_α.
If xy is an edge of H then, by Lemma 4.3, there is a path μ in G'_i from x to y such that $\forall t \in \mu \setminus \{x, y\}$, $label_i(t) \prec label_i(y)$. By Lemma 4.5 b), $\forall t \in \mu \setminus \{x, y\}$, $\alpha^{-1}(t) < \alpha^{-1}(y)$ and, by the Path Lemma, xy is an edge of G^+_α.
Conversely, let xy be an edge of G^+_α. Let us show that xy is an edge of H. By the Path Lemma, there is a path μ in G from x to y such that $\forall t \in \mu \setminus \{x, y\}$, $\alpha^{-1}(t) < \alpha^{-1}(y) < i$, so $\mu \setminus \{x\} \subseteq V_{i-1}$. By Lemma 4.5 c), $\forall t \in \mu \setminus \{x, y\}$, $Num_{i-1}(t) \subseteq Num_{i-1}(y)$. Let t_1 be the neighbor of x in μ. xt_1 is an edge of H, so, by Lemma 4.3, $i \in Num_{i-1}(t_1)$, hence $i \in Num_{i-1}(y)$, and by Lemma 4.3 xy is an edge of H.

We now show that G^+_α is a minimal triangulation of G. Let $H = G^+_\alpha = (V, E + F)$. As G^+_α is a triangulation of G, by the Unique Chord Property, it is sufficient to show that each edge in F is the unique chord of a cycle in H of length 4. Let xy be an edge in F with $\alpha^{-1}(y) < \alpha^{-1}(x) = i$. xy is an edge of H so by Lemma 4.3 there is a path μ in G'_i from x to y such that $\forall t \in \mu \backslash \{x, y\}$, $label_i(t) \prec label_i(y)$, and also $\alpha^{-1}(t) < \alpha^{-1}(y)$ by Lemma 4.5 b). $\mu \setminus \{x, y\} \neq \emptyset$ since xy is not an edge in G. Let t_1 be the vertex in $\mu \setminus \{x, y\}$ such that $\alpha^{-1}(t_1)$ is maximum. By the Path Lemma, xt_1 and t_1y are edges of G^+_α and therefore of H. As $label_i(t_1) \prec label_i(y)$, by Lemma 4.4 $Num_i(y) \not\subseteq Num_i(t_1)$. Let $j \in Num_i(y) \setminus Num_i(t_1)$, and $z = \alpha(j)$. $j > i$ and by Lemma 4.3 yz is an edge of H (and therefore of G^+_α) but t_1z is not. Since yx and yz are edges of G^+_α with $\alpha^{-1}(y) < \alpha^{-1}(x) = i < j = \alpha^{-1}(z)$, by definition of G^+_α xz is an edge of G^+_α, and therefore of H. Hence xy is the unique chord of cycle (x, t_1, y, z, x) in H of length 4. □

Thus MLSM (and also LEX M, MCS-M, LexDFS-M and MNSM) computes an meo and a minimal triangulation of the input graph. An execution of MLSM has the same behaviour (same labeling and numbering) on the input graph G as an execution of MLS on the output minimal triangulation G^+_α, breaking ties in the same way. If moreover G is chordal then G is equal to its minimal triangulation G^+_α, so that MLS and MLSM have the same behaviour on G. We immediately obtain the following two properties:

Property 4.6 *For any graph G and any labeling structure S, any S-MLSM ordering α of G is an S-MLS ordering of G^+_α.*

Property 4.7 *For any chordal graph G and any labeling structure S, G has the same S-MLS and S-MLSM orderings.*

It follows from Property 4.7 and Theorems 3.8 and 4.2 that IC is exactly the condition required on a labeling structure S for S-MLSM to compute only meos of every graph.

Corollary 4.8 *Condition IC imposed on a labeling structure S is a necessary and sufficient condition for S-MLSM to compute only meos of every graph.*

Proof: IC is a sufficient condition by Theorem 4.2.
Conversely, if S-MLSM computes only meos of every graph, then it computes only peos of every chordal graph, and so does S-MLS by Property 4.7. It follows by Theorem 3.8 that S satisfies IC. □

Let us remark that for two given structures S and S', the sets of orderings computed by S-MLSM and S'-MLSM may be different, as is the case for S-MLS and S'-MLS. LEX M and MCS-M for example compute different orderings, as shown in Figure 2 (since MLS and MLSM compute the same orderings on a chordal graph). In the same way that MNS is as general as MLS, it turns out that MNSM is as general as MLSM, thus every graph has the same MLSM and MNSM orderings. The proof goes as for MLS and MNS since by Lemma 4.5 a) $Num_i(y)$ can be defined from the ordering α computed on numbered vertices, independently from the labeling structure used.

Lemma 4.9 *For any graph $G = (V, E)$, any execution of MLSM on G computing some ordering α on V, any integer i between 1 and n and any $y \in V_i$,*

$Num_i(y)) = NumM^\alpha_{G,i}(y))$,

where $NumM^\alpha_{G,i}$ is defined on V_i by induction on i from n down to 1 by:

$NumM^\alpha_{G,n}(y) = \emptyset$, *and for any i from n down to 2,*

$NumM^\alpha_{G,i-1}(y) = NumM^\alpha_{G,i}(y) \cup \{i\}$ *if there is a path μ in G'_i from $\alpha(i)$ to y such that $\forall t \in \mu \setminus \{\alpha(i), y\}$, $NumM^\alpha_{G,i}(t) \subset NumM^\alpha_{G,i}(y)$, otherwise $NumM^\alpha_{G,i-1}(y) = NumM^\alpha_{G,i}(y)$.*

Characterization 4.10 *For any graph G, any labeling structure S, and any ordering α of V, α is an S-MLSM ordering of G if and only if for any integers i, j such that:*
$1 \leq j < i \leq n$, $lab_S(NumM^{\alpha}_{G,i}(\alpha(i))) \not\prec lab_S(NumM^{\alpha}_{G,i}(\alpha(j)))$.

Lemma 4.11 *Let S and S' be labeling structures with partial orders $\preceq_S$ and $\preceq_{S'}$ resp. such that for any subsets I and I' of $\mathbb{Z}_2^+$, if $lab_{S'}(I) \prec_{S'} lab_{S'}(I')$ then $lab_S(I) \prec_S lab_S(I')$.*
Then every S-MLSM ordering of G is also an S'-MLSM ordering of G for every graph G.

Theorem 4.12 *For any graph $G = (V, E)$ and any labeling structure S, any S-MLSM ordering of G is an MNSM ordering of G.*

4.2 The CompMLSM family of algorithms

We define CompMLSM from MLSM in the same way we defined CompMLS from MLS. Properties extend readily from an MLSM algorithm to its CompMLSM version: at Step i, our proofs only compare the label of $\alpha(i)$ to labels of vertices along paths in the graph G'_i, so $\alpha(i)$ needs only be maximal within the connected component of G'_i containing it.

We thus have similar results:

Theorem 4.13 *For any input graph G and any X in {LEX M, MCS-M, LexDFS-M, MNSM, MLSM}, CompX computes a meo α of G and the associated minimal triangulation G^+_α of G.*

We also easily extend results such as Properties 4.6 and 4.7 and Characterization 4.10.

An important difference between MLSM and CompMLSM is that the set of orderings CompMLSM can find is independent of the labeling structure used, and is a superset of the set of orderings obtainable by any algorithm of the MLSM family.

Lemma 4.14 *For any execution of MLSM or CompMLSM on a graph G, any integer i between 1 and n and any vertex y of the connected component of G'_i containing $\alpha(i)$,*
$Num_i(y) \subseteq Num_i(\alpha(i))$.

Proof: Let μ be a path in G'_i between $\alpha(i)$ and y. $\forall t \in \mu \backslash \{\alpha(i)\}$, $\alpha^{-1}(t) < i$, so by Lemma 4.5 c) $Num_i(y) \subseteq Num_i(\alpha(i))$. □

Theorem 4.15 *Any graph has the same S-CompMLSM orderings for all labeling structures S.*

Proof: Let G be a graph and S, S' be labeling structures. Let α be an S-CompMLSM ordering of G, let us show that α is a S′-CompMLSM ordering

of G. By the extension of Characterization 4.10 to CompMLSM, it is sufficient to show that for any integers i, j such that $1 \leq j < i \leq n$ and $\alpha(i)$ and $\alpha(j)$ are in the same connected component of the subgraph of G induced by $\{\alpha(k),\ 1 \leq k < i\}$, $lab_{S'}(NumM^{\alpha}_{G,i}(\alpha(i))) \not\prec lab_{S'}(NumM^{\alpha}_{G,i}(\alpha(j)))$. Let i, j be such integers. By Lemma 4.14 $Num_i(\alpha(j)) \subseteq Num_i(\alpha(i))$ in an execution of S-CompMLSM computing α, so by Lemma 4.9 $NumM^{\alpha}_{G,i}(\alpha(j)) \subseteq NumM^{\alpha}_{G,i}(\alpha(i))$. It follows by condition IC that $lab_{S'}(NumM^{\alpha}_{G,i}(\alpha(j))) \preceq lab_{S'}(NumM^{\alpha}_{G,i}(\alpha(i)))$, and therefore:
$lab_{S'}(NumM^{\alpha}_{G,i}(\alpha(i))) \not\prec lab_{S'}(NumM^{\alpha}_{G,i}(\alpha(j)))$. $\Box$

Every chordal graph G has the same S-CompMLSM and S-CompMLS orderings, which are exactly its peos (Theorem 3.7, whose proof uses Theorem 4.15).

Computing all peos does not extend to meos for the MLSM family of algorithms: Figure 7 shows an meo which CompMLSM is not capable of computing.

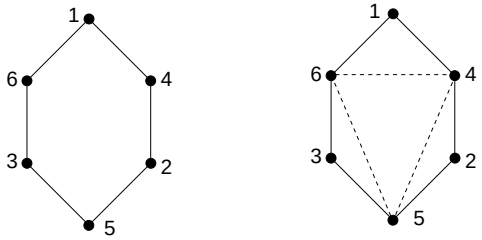

Figure 7: (a) Graph G and an meo α of G. (b) The corresponding minimal triangulation G^{+}_{α} of G. No version of CompMLSM or MLSM can compute this meo, and the corresponding minimal triangulation is not obtainable by any of these algorithms.

This raises the question of which minimal triangulations can be obtained by various algorithms of this family. Villanger in [20] proved the surprising result that the sets of minimal triangulations obtainable by LEX M and MCS-M are the same. Upon investigation, it turns out that, given one of these algorithms, using its Comp version does not enlarge the set of computable triangulations, although the set of computable meos may be larger.

Theorem 4.16 *For any graph G and any given labeling structure S, G has the same sets of S-MLSM and of S-CompMLSM minimal triangulations.*

Proof: Let $G = (V, E)$ be a graph and let S be a labeling structure. Clearly, any S-MLSM minimal triangulation of G is an S-CompMLSM one.

Conversely, let H be an S-CompMLSM minimal triangulation of G and let us show that it is an S-MLSM one. Let α be the ordering on V computed by some execution of S-CompMLSM computing H, and, for any i from 1 to n, let C_i be the connected component of G'_i chosen at step i of this execution. Let α' be the ordering on V and H' be the minimal triangulation

of G computed by an execution of S-MLSM, choosing, for any i from 1 to n, $\alpha'(i)$ at step i in the following way (every variable v is denoted v in the execution of CompMLSM and v' in that of MLSM):
1) Choose a connected component C'_i of G''_i containing a vertex of maximal label in G''_i.
2) If there is some j from 1 to n such that $C'_i = C_j$ and $label'_i(\alpha(j))$ is maximuM in C'_i then choose $\alpha'(i) = \alpha(j)$, otherwise choose $\alpha'(i)$ equal to any vertex of C'_i of maximal label in G''_i.
Note that there is at most one integer j such that $C'_i = C_j$ since for any j, k such that $j < k$, $C_j \neq C_k$ since $\alpha(k) \in C_k \setminus C_j$. Let us show that $H' = H$. For any subset J of $\{1, 2, ..., n\}$, let $\alpha(J)$ denote the set of vertices $\{\alpha(j) \mid j \in J\}$, and, for any i from 1 to n, let $P(i)$ be the following property.
$P(i)$: if there is some j from 1 to n such that $C'_i = C_j$ and $\forall y \in C'_i$, $\alpha'(Num'_i(y)) = \alpha(Num_j(y))$ then the edges of H' produced when processing the vertices of C'_i (in the execution of MLSM) are exactly those of H produced when processing the vertices of C_j (in the execution of CompMLSM).
Let us show $P(i)$ by induction on i from 1 to n. $P(1)$ holds since C'_1 contains a single vertex and processing this vertex produces no edge of H (or H'). We suppose $P(i-1)$ for some i, $1 < i \leq n$. Let us show $P(i)$. We suppose that there is some j from 1 to n such that $C'_i = C_j$ and $\forall y \in C'_i$, $\alpha'(Num'_i(y)) = \alpha(Num_j(y))$. By Lemma 4.14 $\forall y \in C_j$, $Num_j(y) \subseteq Num_j(\alpha(j))$, so $\forall y \in C'_i$, $\alpha'(Num'_i(y)) = \alpha(Num_j(y)) \subseteq \alpha(Num_j(\alpha(j))) = \alpha'(Num'_i(\alpha(j)))$ and therefore $Num'_i(y) \subseteq Num'_i(\alpha(j))$. It follows by Lemma 4.4 that $label'_i(\alpha(j))$ is maximum in C'_i. By definition of α', $\alpha'(i) = \alpha(j)$. The edges of H' produced when processing $\alpha'(i)$ are exactly those of H produced when processing $\alpha(j)$ by Lemma 4.5 a) and the fact that $\forall y \in C'_i$, $\alpha'(Num'_i(y)) = \alpha(Num_j(y))$, and the connected components of G''_{i-1} obtained from C'_i by removing $\alpha'(i)$ are exactly those of G'_{j-1} obtained from C_j by removing $\alpha(j)$ with $\forall y \in C'_i \setminus \{\alpha'(i)\}$, $\alpha'(Num'_{i-1}(y)) = \alpha(Num_{j-1}(y))$. For each such connected component C, there is some $k < i$ and some $l < j$ such that $C = C'_k = C_l$ and $\forall y \in C'_k$, $\alpha'(Num'_k(y)) = \alpha'(Num'_{i-1}(y)) = \alpha(Num_{j-1}(y)) = \alpha(Num_l(y))$, so by induction hypothesis, the edges of H' produced when processing the vertices of C are exactly those of H produced when processing the vertices of C. Hence the edges of H' produced when processing the vertices of C'_i are exactly those of H produced when processing the vertices of C_j. So $P(i)$ holds, which completes the induction on i. Now, for each connected component C of G there are some i and j from 1 to n such that $C = C'_i = C_j$ and $\forall y \in C$, $Num'_i(y) = Num_j(y) = \emptyset$, so by $P(i)$ the edges of H' produced when processing the vertices of C are exactly those of H produced when processing the vertices of C. Hence $H' = H$. $\square$

Theorem 4.16, together with Theorem 4.15, yields the following interesting result:

Theorem 4.17 *For any graph G, whichever meo-computing algorithm of the MLSM and CompMLSM families is used (e.g., LEX M, MCS-M, LexDFS-*

M, MNSM, or their Comp extensions), the set of computable minimal triangulations is the same.

These minimal triangulations fail to cover all possible minimal triangulations: Figure 7(b) shows a minimal triangulation which is obtainable by none of our graph search meo-computing algorithms.

5 Complexity of MLS and MLSM

We now consider the question of implementing Algorithms MLS and MLSM. In the following, we use the word "implementation" in an algorithmic sense and not in a programming one. We will give a detailed version of each algorithm studied in this paper and precise the data structures used in order to compute its time and space complexity, but we will not give any real implementation in a programming language. We will also explore variants Test-MLS and Test-MLSM: Algorithm Test-MLS takes as input a graph $G = (V, E)$, a labeling structure S and an ordering α of V and returns YES if α is an S-MLS ordering of G and NO otherwise. It is obtained from Algorithm MLS by replacing the instructions

Choose a vertex x of G' of maximal label;
$\alpha(i) \leftarrow x$

by

if the label of $\alpha(i)$ is not maximal in G' then return NO
$x \leftarrow \alpha(i)$

and by adding the instruction

return YES

at the end of the algorithm.

Test-MLSM is defined from MLSM in the same way, and we denote by Test-S-MLS, Test-S-MLSM,Test-LexBFS etc the corresponding variants of S-MLS, S-MLSM, LexBFS etc.

An implementation of S-MLS is required to compute only S-MLS orderings, but not to be able to compute every S-MLS ordering of a graph. For instance, as any MCS ordering is a MNS ordering, any implementation of MCS is also an implementation of MNS. However, an implementation finding out if a given ordering is a MCS ordering or not will not be able to find out if this ordering is a MNS ordering or not. An implementation of Test-S-MLS (resp. Test-S-MLSM) will in general have to keep closer to S than S-MLS (resp. S-MLSM). By Lemmas 3.5 and 4.11 we have the following property.

Property 5.1 *Let S and S' be labeling structures with partial orders $\preceq_S$ and $\preceq_{S'}$ resp. such that for any subsets I and I' of $\mathbb{Z}_2^+$, if $lab_{S'}(I) \prec_{S'} lab_{S'}(I')$ then $lab_S(I) \prec_S lab_S(I')$.*
Then any implementation of S-MLS (resp. S-MLSM) is also an implementation of S'-MLS (resp. S'-MLSM).

In particular, S-MLS (resp. S-MLSM) can be implemented by replacing the partial order on labels by anyone of its linear extensions.

5.1 Complexity of MLS and Test-MLS

We will first study the main instances of MLS, namely LexBFS, MCS, LexDFS and MNS, then we will give an implementation of S-MLS with a stratified tree, which is a data structure designed to manipulate priority queues, for any labeling structure S such that labels are positive integers ordered by $\leq$.

An implementation of LexBFS is given by Rose, Tarjan and Lueker [15], and an implementation of MCS is given by Tarjan and Yannakakis [17] with the following complexity results.

Theorem 5.2 *[15, 17] LexBFS and MCS can be implemented in $O(n+m)$ time and space.*

It is easy to check that the data structures used in [15] for LexBFS and in [17] for MCS can be used without extra cost for Test-LexBFS and Test-MCS respectively. Moreover, as by Theorem 3.6 any LexBFS or MCS ordering is an MNS one, we have the following Corollary of Theorem 5.2.

Corollary 5.3 *Test-LexBFS, Test-MCS and MNS can be implemented in $O(n+m)$ time and space.*

We can derive from the implementation of LexBFS given in [15] an implementation of LexDFS.

Implementation of LexDFS and Test-LexDFS with a list of lists.

As in the implementation of LexBFS, the current state of labels is represented by a list L of non-empty lists l. Each list l contains the unnumbered vertices bearing a given label, and the list L is ordered in decreasing order on the labels associated with the lists l (see [15] for a full description of the data structure). It is initialized with a unique list l containing all the vertices of the graph. At Step i, $\alpha(i)$ is chosen in the first list in L and removed from this list, and for each list l in L, the neighbors of $\alpha(i)$ in l are removed from l and put into a new list l_1 which is placed just before l in L. This corresponds to the new decreasing LexBFS order in an execution of LexBFS. To obtain an implementation of LexDFS, it is sufficient to add the following instruction at the end of each Step i: scan the list L to extract the lists l_1 and form a list L_1 with these lists l_1 in the same order, then concatenate L_1 with the remaining list L to obtain the new list L in decreasing LexDFS order.
For Test-LexDFS, we test at iteration i whether $\alpha(i)$ is in the first list of L or not.

Theorem 5.4 *LexDFS and Test-LexDFS can be implemented in $O(n^2)$ time and $O(n+m)$ space.*

Proof: The time complexity of LexDFS is obtained from the time complexity of LexBFS by adding the cost of scanning the list L to form the list

L_1 at each step. As there are n scans and each scan requires $O(n)$ time, we obtain an $O(n^2)$ time complexity for LexDFS. The space complexity is the same as that of LexBFS, i.e. $O(n+m)$. These complexity bounds also hold for Test-LexDFS. □

We will now discuss implementing Test-MNS. We remark that for any vertices v and w of G'_i, the Boolean value of $label_{i-1}(v) \preceq label_{i-1}(w)$ only depends on the value of $label_i(v) \preceq label_i(w)$ and on whether v and w are neighbors of $\alpha(i)$ or not. Thus we can implement Test-MNS by storing these Boolean values instead of storing and comparing explicit labels.

Implementation of Test-MNS.

We use a Boolean matrix $Preceq$ such that at the beginning of Step i, for any vertices v and w of G'_i, $Preceq(v,w) = True$ iff $label_i(v) \preceq label_i(w)$. $Preceq$ is initialized with True. Testing the maximality of the label of $\alpha(i)$ in G' is implemented by testing the absence of a vertex v of G' such that $Preceq(\alpha(i), v)$ and not $Preceq(v, \alpha(i))$. Labels are updated at Step i by the following procedure, where $x = \alpha(i)$:

foreach neighbor v of x in G' **do**
  foreach non-neighbor w of x in G' **do**
    $Preceq(v,w) \leftarrow False$

It is easy to check that this procedure correctly updates matrix $Preceq$ with respect to its desired meaning, so that this implementation of Test-MNS is correct.

Theorem 5.5 *Test-MNS can be implemented in $O(n(n+m))$ time and $O(n^2)$ space.*

Proof: Initialization requires $O(n^2)$ time and at each step i, testing the maximality of the label of $\alpha(i)$ in G' requires $O(n)$ time and updating matrix $Preceq$ requires $O(n|N(\alpha(i))|)$ time, which makes a global $O(n(n+m))$ time bound. Matrix $Preceq$ requires $O(n^2)$ space, which is the space bound of the algorithm. □

Note that we can derive from this implementation of Test-MNS an implementation of MNS which is able to compute every MNS ordering of the input graph with the same time and space bounds. In addition to matrix $Preceq$, we use an array containing for each vertex v of G' the number of vertices w of G' having a larger label than v (i.e. such that $Preceq(v,w)$ and not $Preceq(w,v)$). This array allows to choose at each step a vertex of maximal label in G' in $O(n)$ time.

5.1.1 Using a stratified tree

For any labeling structure S such that labels are positive integers ordered by $\leq$, S-MLS can be implemented with the data structure of a stratified tree defined by Van Emde Boas [18, 19] to manipulate priority queues. This

data structure is used to implement a subset C of an interval of integers in the form $[1, n]$ ordered by $\leq$, which here will be the set of current labels, i.e. the set of labels assigned to unnumbered vertices at some point of the execution of S-MLS. The stratified tree can be initialized in $O(n \log\log n)$ time. Inserting or removing an element, or finding the maximum element in C requires $O(\log\log n)$ time. The structure requires $O(n)$ space. These bounds are computed in the model of the unit-cost RAM.

Implementation of S-MLS and Test-S-MLS with a stratified tree.

We suppose that labels are positive integers in some interval I ordered by $\leq$. The set C of current labels is stored in a stratified tree. With each current label is associated the non-empty list of unnumbered vertices having this label. These lists can be stored in an array indexed by the integers in I. At the initialization step, the unique element l_0 of C is associated with the list of all vertices of the graph. To choose an unnumbered vertex with maximal label at Step i, we find the maximum element l_{max} of C, remove a vertex from the list associated with l_{max} and remove l_{max} from C if this list has become empty. For Test-S-MLS, it is tested whether $\sigma(i)$ has label l_{max} or not. To update the label of a vertex v from l to l', we transfer v from the list associated with l to the list associated with l', with a possible removal of l from C and a possible insertion of l' into C.

Proposition 5.6 *Let S be a labeling structure such that for any integer $n \geq 1$, set $L_n = \{lab_S(I),\ I \subseteq [1, n]\}$ is a subset of an interval $[1, r(n)]$ of integers ordered by $\leq$, with $r(n)$ in $O(n)$.*
Then S-MLS and Test-S- MLS can be implemented in $O((n+m)\log\log n + m\, t_{Inc}(n))$ time and $O(n+m)$ space, where $t_{Inc}(n))$ is the time required to increment a label of L_n.

Proof: Implementing the set C of current labels with a stratified tree requires $O(n \log\log n)$ initialization time, $O(\log\log n)$ time per operation and $O(n)$ space [18, 19]. As choosing a vertex with maximal label and updating the label of a vertex require at most a constant number of operations on the stratified tree, we obtain the announced bounds. □

As for CompMLS, for any labeling structure S, S-CompMLS can be implemented by any implementation of S-MLS since any S–MLS ordering is an S-CompMLS one. Moreover, by Theorem 3.7 S-CompMLS restricted to chordal graphs can be implemented by an implementation of LexBFS or MCS. Test-S-CompMLS restricted to chordal graphs only has to determine if the given ordering is a peo of the graph or not, which can be implemented in linear time and space [15].

5.2 Complexity of MLSM and Test-MLSM

MLSM is more complex than MLS since graph G' must additionally be searched at each Step i to determine the vertices whose label has to be

incremented, i.e. the neighbors of $\alpha(i)$ in the minimal triangulation H of G. We will show that this search can be performed using another labeling structure than the input labeling structure S, which will allow us to deduce an implementation and complexity bounds of S-MLSM from those of LEX M and S-MLS.

An implementation of LEX M is given by Rose, Tarjan and Lueker [15] with the following complexity results, where m' denotes the number of edges of the computed minimal triangulation H of G.

Theorem 5.7 *[15] LEX M can be implemented in $O(n(n+m))$ time and $O(n+m')$ space.*

Note that the implementation of LEX M given in [15] only requires $O(n+m)$ space because it only computes an meo α of G and not its minimal triangulation $H = G^+_\alpha$. We will show that this implementation of LEX M can be used to implement S-MLS for any labeling structure S. We first extend Lemma 4.5 a) to the following lemma:

Lemma 5.8 *For any graph G, any labeling structures S and S' with partial orders $\preceq_S$ and $\preceq_{S'}$ resp., any execution of MLSM on G and S, any integer i between 1 and n and any path μ in G'_i ending in some vertex y,*

$\forall t \in \mu \setminus \{y\}$, $label_i(t) \prec_S label_i(y)$ iff $\forall t \in \mu \setminus \{y\}$, $lab_{S'}(Num_i(t)) \prec_{S'} lab_{S'}(Num_i(y))$.

Proof: By Lemma 4.5 a) it is sufficient to show that:
$\forall t \in \mu\setminus\{y\}$, $lab_{S'}(Num_i(t)) \prec_{S'} lab_{S'}(Num_i(y))$ iff $\forall t \in \mu\setminus\{y\}$, $Num_i(t) \subset Num_i(y)$. The proof of this last equivalence is similar to that of Lemma 4.5 a), replacing the references to Lemma 4.4 with references to condition IC. □

We distinguish in an execution of MLSM the *specific MLSM part* which is the search in G' at each Step i from $\alpha(i)$ to determine the vertices whose label has to be incremented, from the *MLS part* which corresponds to an execution of MLS on the output graph G^+_α. We suppose in the following result that the MLS part of MLSM on G can be implemented with the same time and space complexity as MLS on G^+_α. In other words, we assume that the fact that G^+_α is only partially known at each step of an execution of MLSM does not affect the complexity of MLS, which seems reasonable since the unknown edges of G^+_α are between unnumbered edges and therefore play no role in the algorithm.

Theorem 5.9 *For any labeling structure S, if S-MLS (resp. Test-S-MLS) can be implemented in $O(f(n,m))$ time and $O(g(n,m))$ space then S-MLSM (resp. Test-S-MLSM) can be implemented in $O(n(n+m)+f(n,m'))$ time and $O(g(n,m'))$ space.*

Proof: We implement the specific MLSM part of Algorithm S-MLSM or Test-S-MLSM with the part of the implementation of LEX M given in [15]

corresponding to the specific MLSM part and the updating of the integer labels $l(v)$ at each Step. As the order on these labels $l(v)$ at the beginning of Step i is the same as the lexicographic order on the $lab_{S_1}(Num_i(v))$ with S_1-MLSM = LEX M [15], Lemma 5.8 ensures that this correctly implements the specific MLSM part of the algorithm. By our assumption, the MLS part of S-MLSM (resp. Test-S-MLSM) can be implemented by an implementation of S-MLS (resp. Test-S-MLS) with the same complexity as this algorithm on G_α^+. The result follows from Theorem 5.7. □

Corollary 5.10 *MCS-M, LexDFS-M, MNSM, Test-LEX M, Test-MCS-M and Test-LexDFS-M can be implemented in $O(n(n+m))$ time and $O(n+m')$ space.*
Test-MNSM can be implemented in $O(n(n+m'))$ time and $O(n^2)$ space.

For some labeling structures S, we can directly derive from the implementation of LEX M given in [15] an implementation of S-MLSM by just modifying the way labels $l(v)$ are updated to make them correspond to the labels obtained with the labeling structure S. For instance, we obtain an implementation of LexDFS-M by replacing the instruction $l(z) := l(z)+1/2$ by $l(z) := l(z)+n$, and we obtain an implementation of MCS-M by replacing this instruction by $l(z) := l(z) + 1$ and by replacing the procedure *sort* by: set k to the maximum value of $l(v)$ for unnumbered vertices v. This implementation of MCS-M is a little simpler than the implementation of LEX M (with the same complexity bounds) since it avoids renaming all labels in the procedure *sort*. It can be used in the implementation of S-MLSM and Test-S-MLSM instead of the implementation of LEX M, and is itself an implementation of MNSM by Theorem 4.12. In the same way, we can define direct implementations of Test-LexDFS-M and Test-MCS-M, but not of Test-MNSM. If labels are positive integers ordered by $\leq$, we can implement the MLS part with a stratified tree, and deduce complexity bounds from Proposition 5.6 and Theorem 5.9. By Theorem 4.15, for any labeling structure S, S-CompMLSM can be implemented by an implementation of LEX M or MCS-M.

6 Conclusion

We have extended Algorithm LexBFS into Algorithm MLS by defining a general labeling structure, and shown how to extend this further to CompMLS to enable any possible peo to be computed. We have also extended all these algorithms to meo-computing versions. Our work yields alternate (and often simpler) proofs for the results of several papers, as [1, 15, 16, 17, 20].

However, we have shown that these new meo-computing algorithms fail to enhance the possibility for finding a wider range of minimal triangulations. LEX M has been studied experimentally, and shown to be very restrictive ([5]), yielding triangulations which for example are far from edge-number

minimum. This problem remains with the enlarged family of new meo-computing algorithms we present here, and appears to be a fundamental limitation of graph search.

We presented time and space complexity bounds of some algorithms of the MLS and MLSM families. These results mostly derive from the known complexity bounds of LexBFS, MCS and LEX M. An interesting fact is that the search in G' at each step of an execution of MLSM can be performed using any other labeling structure than the input labeling structure S, which allows to implement S-MLSM by combining implementations of LEX M and S-MLS.

As mentioned in the Introduction, LexBFS and MCS, though designed for chordal graphs, have been used for graph classes other than chordal graphs. The more general peo-finding algorithms discussed in this paper could also prove useful on non-chordal graphs, on a wider variety of graph classes and problems.

References

[1] A. Berry, J. Blair, P. Heggernes and B. Peyton. Maximum Cardinality Search for computing minimal triangulations of graphs. *Algorithmica*, 39(4):287–298, 2004.

[2] A. Berry and J.-P. Bordat. Separability generalizes Dirac's theorem. *Discrete Applied Mathematics*, 84:43–53, 1998.

[3] A. Berry and J.-P. Bordat. Moplex elimination orderings. *Electronic Notes in Discrete Mathematics, Proceedings of First Cologne-Twente Workshop on Graphs and Combinatorial Optimization* , 8, 2001.

[4] A. Berry, R. Krueger, and G. Simonet. Ultimate generalizations of LexBFS and LEX M. *Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science 2005 (WG 2005)*, LNCS 3787, Springer-Verlag, 199–213, 2005.

[5] J. R. S. Blair, P. Heggernes, and J. A. Telle. A practical algorithm for making filled graphs minimal. *Theoretical Computer Science A*, 250-1/2: 125–141, 2001.

[6] H. L. Bodlaender, A. M. C. A. Koster. On the Maximum Cardinality Search Lower Bound for Treewidth. *Proceedings (WG 2004)*, 81–92, 2004.

[7] H. L. Bodlaender, A. M. C. A. Koster. On the Maximum Cardinality Search Lower Bound for Treewidth. *Discrete Applied Mathematics* 155: 1348–1372, 2007.

[8] D. G. Corneil "Lexicographic Breadth First Search - a survey *30th International Workshop on Graph Theory (WG2004)*, LNCS 3353; Springer, 1–19, 2004.

[9] D. G. Corneil and R. Krueger. A unified view of graph searching. Submitted.

[10] D. G. Corneil, S. Olariu, and L. Stewart. Linear Time Algorithms for Dominating Pairs in Asteroidal Triple-free Graphs. *SIAM Journal on Computing*, 28:1284–1297, 1999.

[11] E. Dahlhaus, P. L. Hammer, F. Maffray, and S. Olariu. On Domination Elimination Orderings and Domination Graphs. *Proceedings of WG 1994*, 81–92, 1994.

[12] D.R. Fulkerson and O.A. Gross. Incidence matrixes and interval graphs. *Pacific Journal of Math.*, 15:835–855, 1965.

[13] T. Ohtsuki. A fast algorithm for finding an optimal ordering in the vertex elimination on a graph. *SIAM Journal on Computing*, 5:133–145, 1976.

[14] T. Ohtsuki, L. K. Cheung, and T. Fujisawa. Minimal triangulation of a graph and optimal pivoting order in a sparse matrix. *Journal of Math. Analysis and Applications*, 54:622–633, 1976.

[15] D. Rose, R.E. Tarjan, and G. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journ. Comput*, 5:146–160, 1976.

[16] D. R. Shier. Some aspects of perfect elimination orderings in chordal graphs. *Discrete Applied Mathematics*, 7:325–331, 1984.

[17] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13:566–579, 1984.

[18] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inform. Process. Lett.*, 6(3):80–82, 1977.

[19] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Math. Systems Theory*, 10:99–127, 1977.

[20] Y. Villanger. Lex M versus MCS-M. *Discrete Mathematics*, 306(3): 393-400, 2004.

Appendix

We give the proof of Lemma 4.5.

Lemma 4.5 *For any graph G, any execution of MLSM on G computing ordering α, any integer i from 1 to n and any path μ in G'_i ending in some vertex y,*

a) $\forall t \in \mu \setminus \{y\},\ label_i(t) \prec label_i(y)$ *iff* $\forall t \in \mu \setminus \{y\},\ Num_i(t) \subset Num_i(y)$,

b) if $\forall t \in \mu \setminus \{y\},\ label_i(t) \prec label_i(y)$ *then* $\forall t \in \mu \setminus \{y\},\ \alpha^{-1}(t) < \alpha^{-1}(y)$,

c) if $\forall t \in \mu \setminus \{y\},\ \alpha^{-1}(t) < \alpha^{-1}(y)$ *then* $\forall t \in \mu \setminus \{y\},\ Num_i(t) \subseteq Num_i(y)$.

The proof uses the following technical Lemmas 6.1 and 6.2. For any path μ containing vertices x and y, $\mu[x, y]$ denotes the subpath of μ between x and y.

Lemma 6.1 *For any graph G, any execution of MLSM on G, any integer i from 1 to n and any path μ in G'_{i-1} ending in some vertex y,*
if $\forall t \in \mu \setminus \{y\},\ Num_i(t) \subset Num_i(y)$ *then* $\forall t \in \mu \setminus \{y\},\ Num_{i-1}(t) \subset Num_{i-1}(y)$.

Proof: We suppose that $\forall t \in \mu \setminus \{y\},\ Num_i(t) \subset Num_i(y)$ (and therefore $label_i(t) \prec label_i(y)$ by Lemma 4.4). Let $t \in \mu \setminus \{y\}$ and let us show that $Num_{i-1}(t) \subset Num_{i-1}(y)$. It is sufficient to show that if $i \in Num_{i-1}(t)$ then $i \in Num_{i-1}(y)$. We suppose that $i \in Num_{i-1}(t)$. By Lemma 4.3 there is a path λ in G'_i from $\alpha(i)$ to t such that $\forall t' \in \lambda \setminus \{\alpha(i), t\},\ label_i(t') \prec label_i(t)$. Let μ' be the path obtained by concatenation of λ and $\mu[t, y]$. Then μ' is a path in G'_i from $\alpha(i)$ to y such that $\forall t' \in \mu' \setminus \{\alpha(i), y\},\ label_i(t') \prec label_i(y)$. Hence by Lemma 4.3 $i \in Num_{i-1}(y)$. □

Lemma 6.2 *For any graph G, any execution of MLSM on G, any integer i from 1 to n and any path μ in G'_i ending in some vertex y,*
if $\exists t \in \mu \setminus \{y\} \mid Num_i(t) \not\subseteq Num_i(y)$ *then* $\exists t_1 \in \mu \setminus \{y\} \mid \forall t \in \mu[t_1, y] \setminus \{t_1\},\ Num_i(t) \subset Num_i(t_1)$.

Proof: We suppose that $\exists t \in \mu \setminus \{y\} \mid Num_i(t) \not\subseteq Num_i(y)$. Let j be the largest integer such that $\exists t \in \mu \setminus \{y\} \mid Num_{j-1}(t) \not\subseteq Num_{j-1}(y)$ and let t_1 be the vertex of μ closest to y such that $Num_{j-1}(t_1) \not\subseteq Num_{j-1}(y)$. So $j - 1 \geq i$, $j \in Num_{j-1}(t_1)$ and $\forall t \in \mu[t_1, y] \setminus \{t_1\},\ j \notin Num_{j-1}(t)$. Let us show that $Num_j(t_1) = Num_j(y)$. We assume for contradiction that $Num_j(t_1) \neq Num_j(y)$. Let t_2 be the vertex of $\mu[t_1, y]$ closest to t_1 such that $Num_j(t_2) = Num_j(y)$. By the choice of j, $\forall t \in \mu[t_1, t_2] \setminus \{t_2\},\ Num_j(t) \subset Num_j(t_2)$ and by Lemma 6.1 $Num_{j-1}(t_1) \subset Num_{j-1}(t_2)$. So $j \in Num_{j-1}(t_2)$ with $t_2 \in \mu[t_1, y] \setminus \{t_1\}$, a contradiction.
So $\forall t \in \mu[t_1, y] \setminus \{t_1\},\ Num_{j-1}(t) = Num_j(t) \subseteq Num_j(y) = Num_j(t_1) \subset$

$Num_j(t_1) \cup \{j\} = Num_{j-1}(t_1)$. As $j - 1 \geq i$, by Lemma 6.1 $\forall t \in \mu[t_1, y] \setminus \{t_1\},\ Num_i(t) \subset Num_i(t_1)$. $\square$

Proof: (of Lemma 4.5) a) For the forward direction, we assume for contradiction that $\forall t \in \mu \setminus \{y\},\ label_i(t) \prec label_i(y)$ and $\exists t \in \mu \setminus \{y\} \mid Num_i(t) \not\subset Num_i(y)$ (and therefore $Num_i(t) \not\subseteq Num_i(y)$ since, by Lemma 4.4 (i), $Num_i(t) \neq Num_i(y)$). By Lemma 6.2, $\exists t_1 \in \mu \setminus \{y\} \mid Num_i(y) \subset Num_i(t_1)$ and by Lemma 4.4 $label_i(y) \prec label_i(t_1)$, a contradiction.
The reverse direction follows immediately from Lemma 4.4.
b) We suppose that $\forall t \in \mu \setminus \{y\},\ label_i(t) \prec label_i(y)$.
Let $k = max\{\alpha^{-1}(t),\ t \in \mu\}$. By a) and Lemma 6.1, $\forall t \in \mu \setminus \{y\},\ label_k(t) \prec label_k(y)$. So $\alpha(k) = y$, which completes the proof.
c) We assume for contradiction that $\forall t \in \mu \setminus \{y\},\ \alpha^{-1}(t) < \alpha^{-1}(y)$ and $\exists t \in \mu \setminus \{y\} \mid Num_i(t) \not\subseteq Num_i(y)$. By Lemma 6.2 $\exists t_1 \in \mu \setminus \{y\} \mid \forall t \in \mu[t_1, y] \setminus \{t_1\},\ Num_i(t) \subset Num_i(t_1)$ and by a) and b), $\alpha^{-1}(y) < \alpha^{-1}(t_1)$, a contradiction, $\square$