A simple algorithm to generate the minimal separators and the maximal cliques of a chordal graph

Anne Berry, Romain Pogorelcnik

A simple algorithm to generate the minimal separators and the maximal cliques of a chordal graph

Anne Berry[1] Romain Pogorelcnik[1]

Research Report LIMOS/RR-10-04

February 11, 2010

[1]LIMOS UMR CNRS 6158, Ensemble Scientifique des Cézeaux, F-63 173 Aubière, France, berry@isima.fr

Abstract

We present a simple unified algorithmic process which uses either LexBFS or MCS on a chordal graph to generate the minimal separators and the maximal cliques in linear time in a single pass.

Keywords: MCS, LexBFS, minimal separator, maximal clique, chordal graph.

1 Introduction

Generating the minimal separators and the maximal cliques of a chordal graph is simple with the help of specific search algorithms.

Chordal graphs (which are the graphs with no chordless cycle on four or more vertices), were characterized by Fulkerson and Gross [7] as the graphs for which one can repeatedly find a simplicial vertex (*i.e.* a vertex whose neighborhood is a clique) and remove it from the graph, until no vertex is left; this process, called *simplicial elimination*, defines an ordering α on the vertices called a *perfect elimination ordering (peo)*. ($\alpha(i)$ denotes the vertex bearing number i, and $\alpha^{-1}(x)$ the number of vertex x). At each step of the elimination process, a new *transitory graph* is defined.

Rose [9] showed that for any given peo α of a chordal graph, any minimal separator of the graph is defined in the course of the simplicial elimination process as the transitory neighborhood of some vertex. Moreover, it is easy to see that in any maximal clique K, the vertex y of smallest number by α defines K as its transitory closed neighborhood.

In the rest, we will call such vertices *generators*, and our goal will be to detect these generators efficiently. We will use graph search algorithms LexBFS [10] and MCS [11], both tailored to generate a peo of a chordal graph.

The idea behind this is quite simple. Both algorithms number the vertices from n to 1 (n is the number of vertices of the graph). Each vertex bears a label, which may be modified as the algorithm proceeds. At each step, a new vertex is chosen to be numbered. If the label of this new vertex x_i is not larger that the label of the previous vertex x_{i+1}, then we know that x_i generates a minimal separator of the graph. The vertices of this minimal separator are the already numbered neighbors of x_i. We also know that the previously numbered vertex, x_{i+1}, generates a maximal clique: x_{i+1}, together with its already numbered neighbors (excluding x_i), define a maximal clique of the graph.

Thus the labels of algorithms LexBFS and MCS enable the user to detect these generators as soon as they are numbered. The minimal separators and maximal cliques can be found "on-line", without requiring a preliminary pass of the algorithm to number all the vertices.

The corresponding results have been proved for MCS in two separate papers. Blair and Peyton [5], while studying how MCS defines a clique tree of a chordal graph, showed how to generate the maximal cliques (which are the nodes of the clique tree), using the MCS labels. Kumar and Madhavan [8] showed how to generate the minimal separators of a chordal graph, given an MCS ordering.

However, although a minimal separator and a maximal clique generator are actually detected at the same step of the algorithm, these results have not been unified. Moreover, [8] use a peo as input, but the minimal separators as well as the maximal cliques can be computed during the execution, which may be an important feature when handling very large graphs, since a global pre-numbering of the vertices is not necessary or even useful.

LexBFS, as we will show, exhibits the same property as MCS regarding these generators. Our aim is to present a simple unified single-pass algorithm which generates the minimal separators and the maximal cliques of a chordal graph.

The paper is organized as follows: in Section 2, we give some preliminary results and definitions. In Section 3, we present our main theorem, which describes how the generators are detected, and discuss its proof for LexBFS. Section 4 presents our algorithm and provides an example.

2 Preliminaries

In the rest, we will consider all graphs to be connected (for a disconnected graph, the processes we describe can be applied independently to each connected component).

A graph is denoted $G = (V, E)$, $|V| = n, |E| = m$. Vertices x and y are *adjacent* if $xy \in E$. For $X \subset V$, $G(X)$ denotes the *subgraph induced* by X. The *neighborhood* $N_G(x)$ of vertex x in graph G is $N_G(x) = \{y \neq x | xy \in E\}$ (subscripts will be omitted when it is clear which graph we work on). The *closed neighborhood* is $N_G[x] = \{x \cup N(x)\}$. The neighborhood of a vertex set $X \subset V$ is $N_G(X) = \cup_{x \in X} N_G(x) - X$. A subset X of vertices is called a *module* if the vertices of X share the same external neighborhood: $\forall x \in X, N(x) - X = N(X)$. A *clique* is a set of pairwise adjacent vertices.

A subset S of vertices of a connected graph G is called a *separator* if $G(V - S)$ is not connected. A separator S is called an *xy-separator* if x and y lie in different connected components of $G(V - S)$, a *minimal xy-separator* if S is an xy-separator and no proper subset of S is an xy-separator. A separator S is a *minimal separator*, if there is some pair $\{x, y\}$ such that S is a minimal xy-separator. Alternately, S is an minimal separator if and only if $G(V - S)$ has at least 2 connected components C_1 and C_2 such that $N(C_1) = N(C_2) = S$; such components are called *full components* of S in G, and S is then a minimal xy-separator for any $\{x, y\}$ with $x \in C_1$ and $y \in C_2$.

LexBFS [10] and MCS [11] are both linear-time search algorithms which number the vertices from n to 1, thereby defining a peo. Each vertex x bears a label which corresponds to the list of numbers of the neighbors of x with

a higher number than that of x (LexBFS) or to the cardinality of this list (MCS). Both algorithms are recalled below.

Algorithm LexBFS (Lexicographic Breadth-First Search)[10]

input : A graph $G = (V, E)$.
output: An ordering α of V.
Initialize all labels as the empty string;
for $i \leftarrow n$ **to** 1 **do**
    Pick an unnumbered vertex v whose label is maximal under lexicographic order;
    $\alpha(i) \leftarrow v$;
    foreach *unnumbered vertex* w *adjacent to* v **do**
        append i to $label(w)$;

Algorithm MCS (Maximal Cardinality Search)[11]

input : A graph $G = (V, E)$.
output: An ordering α of V.
Initialize all labels as 0;
for $i \leftarrow n$ **to** 1 **do**
    Pick an unnumbered vertex v with maximum label;
    $\alpha(i) \leftarrow v$;
    foreach *unnumbered vertex* w *adjacent to* v **do**
        $label(w) \leftarrow label(w) + 1$;

3 Main Theorem

We will now present our main theorem, from which our algorithm is derived. We will need the following definitions:

Definition 3.1 *Given a chordal graph G and a peo α of G, for any vertex x, $Madj(x)$ is the set of neighbors of x with a number higher than that of x: $Madj(x) = \{y \in N(x) | \alpha^{-1}(y) > \alpha^{-1}(x)\}$.*

Definition 3.2 *Given a chordal graph G and a peo α of G, we will call:*

- minimal separator generator *any vertex x_i with number i by α, such that $Madj(x_i)$ is a minimal separator of G and $label(x_i) \leq label(x_{i+1})$, where $x_{i+1} = \alpha(i+1)$.*
- maximal clique generator *a vertex y such that $Madj(y) \cup \{y\}$ is a maximal clique of G.*

Our main result is the following:

Theorem 3.3 *Let α be a peo defined by either LexBFS or MCS, let x_i be the vertex of number i, let x_{i+1} be the vertex of number $i+1$.*

a) x_i is a minimal separator generator if and only if $label(x_i) \leq label(x_{i+1})$.

b) x_{i+1} is a maximal clique generator if and only if $label(x_i) \leq label(x_{i+1})$ or $i = 1$.

We will now discuss the proof of Theorem 3.3. We will first present moplex elimination, which is a process that explains how both MCS and LexBFS scan the minimal separators and the maximal cliques of a chordal graph, then prove Theorem 3.3 for LexBFS.

3.1 Moplex elimination

We will use the notion of *moplex*, introduced by [1]:

Definition 3.4 *[1] A moplex is a clique X such that X is a module and $N(X)$ is a minimal separator. We extend this definition to a clique whose neighborhood is empty. A simplicial moplex is a moplex whose vertices are all simplicial.*

[1] proved:

Property 3.5 *[1] Any chordal graph which is not a clique has at least two non-adjacent simplicial moplexes.*

From this, they derived a variant of the characterization of Fulkerson and Gross for chordal graphs by simplicial elimination of vertices:

Characterization 3.6 *[1] A graph is chordal if and only if one can repeatedly delete a simplicial moplex until the graph is a clique (which we will call the 'terminal moplex'). We call this process* moplex elimination.

Note that moplex elimination on a chordal graph is a special case of simplicial elimination, since at each step a set of simplicial vertices is eliminated. Note also that for a connected graph G, the transitory elimination graph obtained at the end of each step remains connected.

Moplex elimination defines an ordered partition $(X_1, X_2, ..., X_k)$ of the vertices of the graph into the successive moplexes which are defined in the successive transitory elimination graphs. We will call this partition a *moplex ordering*.

Our basic result is the following:

Theorem 3.7 [1] *Let G be a chordal graph, let $(X_1, X_2, ..., X_k)$ be a moplex ordering of G. At each step $i < k$ of the elimination process finding moplex X_i in transitory graph G_i,*

- *$N_{G_i}(X_i)$ is a minimal separator of G.*
- *$X_i \cup N_{G_i}(X_i)$ is a maximal clique of G.*

The terminal moplex X_k is a maximal clique.
There are no other minimal separators or maximal cliques in G.

To prove this, we will first recall the result from Rose [9]:

Property 3.8 *[9] Let G be a chordal graph, let α be a peo of G, let S be a minimal separator of G. Then there is some vertex x such that $Madj(x) = S$.*

From Property 3.8, it is easy to deduce the following well-known property:

Property 3.9 *[9] Let G be a chordal graph, let S be a minimal separator of G. Then in every full component C of S, there is some vertex x such that $S \subseteq N(x)$ (such a vertex is called a* confluence point *of C).*

Let us now prove Theorem 3.7.

Proof: (of Theorem 3.7) Let us first prove that $N_{G_i}(X_i)$ is a minimal separator of G.

In transitory graph G_i, $N_{Gi}(X_i)$ is by definition a minimal separator S_i of G_i, with at least two full components C_1 and C_2, each containing a confluence point, which we will call x_1 and x_2. Suppose S is not a minimal separator of G. Then there must be a chordless path from x_1 to x_2 in G which contains no vertex of S_i. Let y be the first vertex of this path to be eliminated, at

[1] The results proved in this subsection were briefly presented in [2].

step $j < i$; y must be simplicial in G_j, but this is impossible, since y has two non-adjacent neighbors on the path.

Thus every $N_{G_i}(X_i)$ $(i < k)$ is a minimal separator of G, and by Property 3.8, all the minimal separators of G have thus been encountered.

Let us now prove that $X_i \cup N_{G_i}(X_i)$ is a maximal clique of G, by induction on i.

Clearly, $X_1 \cup N_G(X_1)$ is a maximal clique of G. Let us examine $X_i \cup N_{G_i}(X_i)$: it is a maximal clique of G_i and thus a clique of G. Suppose it is not a maximal clique of G, and let y be a vertex belonging to a larger clique of G containing $X_i \cup N_{G_i}(X_i)$. Since $N_{Gi}(X_i)$ is a minimal separator of G, y must belong to the same full component of $N_{Gi}(X_i)$ as the vertices of X_i. But then y belongs to the same maximal clique module as X_i, a contradiction.

The terminal moplex X_k is by definition a clique; suppose it is not a maximal clique: then it can be augmented with some vertex y, which belongs to some moplex X_i. By Theorem 3.7, $S = N_{Gi}(X_i)$ is a minimal separator of G. Let $C_1, ..., C_t$ be the full components of S in G, and let z be the confluence point belonging to the moplex of highest number X_j. In G_j, S cannot be a minimal separator, since all the other full components of S have been eliminated from G_j. Therefore, $X_j \cup N_{Gj}(X_j)$ must be the terminal moplex, a contradiction. □

Note that for a given chordal graph G, there may be many different moplex orderings, but there is always the same number of moplexes, since this is the number of maximal cliques of the graph.

With any moplex ordering $(X_1, X_2, ..., X_k)$, we can associate a peo α by processing the moplexes from 1 to n and giving consecutive numbers to the vertices of a given moplex. Using α, we can define the minimal separators and maximal cliques as vertex neighborhoods:

Property 3.10 *Let G be a chordal graph, let $(X_1, X_2, ..., X_k)$ be a moplex ordering of G, let α be a peo associated with this moplex ordering. Then in the course of a moplex elimination, at each step i, processing moplex X_i,*

- *The vertex x of moplex X_i whose number is the smallest by α defines a maximal clique $\{x\} \cup Madj(x)$ of G.*
- *The vertex y of moplex X_i whose number is the highest by α defines a minimal separator $Madj(y)$ of G.*

3.2 Generators of LexBFS

LexBFS defines a moplex ordering and an associated peo: [1] showed that LexBFS always numbers as 1 a vertex belonging to a moplex (which we will call X_1). They also proved that the vertices of X_1 receive consecutive numbers by LexBFS. These properties are true at each step of LexBFS in the transitory elimination graph. Therefore, LexBFS defines a moplex elimination $(X_1, ..., X_k)$, by numbering consecutively the vertices of X_1, then numbering consecutively the vertices of X_2, and so forth:

Theorem 3.11 *In a chordal graph, LexBFS defines a moplex ordering.*

Note that it is easy to deduce from [5] that MCS also defines a moplex ordering.

Example 3.12 *Figure 1 shows the numbers and labels of a LexBFS execution on a chordal graph.*

moplex	*minimal separator*	*maximal clique*
$X_1 = \{g\}$	$\{b, c\}$	$\{b, c, g\}$
$X_2 = \{a\}$	$\{b, c, d\}$	$\{a, b, c, d\}$
$X_3 = \{d, h\}$	$\{b, c\}$	$\{b, c, d, h\}$
$X_4 = \{c\}$	$\{b, f\}$	$\{b, c, f\}$
$X_5 = \{b, e, f\}$	-	$\{b, e, f\}$

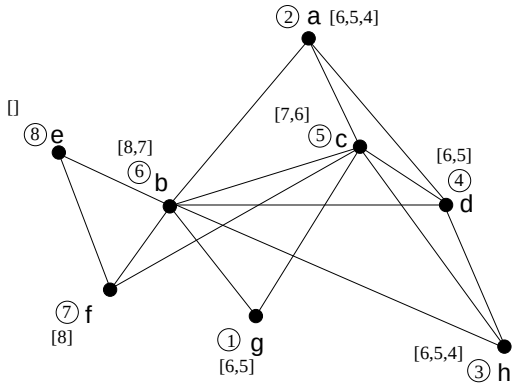

Figure 1: Numbers and labels of a LexBFS execution on a chordal graph.

Since the vertices of any transitory moplex X_i are numbered consecutively, the labels will increase w.r.t. lexicographic order until the entire moplex has been numbered, and then will stop increasing. More precisely, when

running LexBFS (numbering the vertices from n to 1), as long as the labels increase, we are defining a moplex, X_i. When the labels stop increasing (when numbering vertex y), then we have started a new moplex X_{i-1} which will contain y. Using Property 3.10, we can deduce that LexBFS generates the minimal separators and the maximal cliques according to Theorem 3.3.

Thus it is easy, using a LexBFS ordering, to define with a single pass

- the minimal separators of G
- the maximal cliques of G
- the corresponding moplex ordering

4 Algorithm

We can now derive from Theorem 3.3 a generalized algorithm to generate the minimal separators and the maximal cliques of a chordal graph.

In the algorithm below, it is considered that either LexBFS or MCS is used. "Use i to increment the label of y" is translated as "add i to label of y" for LexBFS and as "add 1 to the label of y" for MCS. The labels are all considered initialized at the beginning, as $\emptyset$ for LexBFS and as 0 for MCS.

$G_{NUM} \leftarrow G_{NUM}+\{x_i\}$ is shorthand for "$V_{NUM} \leftarrow V_{NUM}+\{x_i\}$; $G_{NUM} \leftarrow G(V_{NUM})$". In the same fashion, $G_{ELIM} \leftarrow G_{ELIM} - \{x_i\}$ is shorthand for $V_{ELIM} \leftarrow V_{ELIM} - \{x_i\}$; $G_{ELIM} \leftarrow G(V_{ELIM})$;

Symbol + denotes disjoint union.

Note that for LexBFS, if x_i is a minimal separator generator, then the label of x_i gives the separator directly. If, for instance, $label(x_i) = (9, 7, 6)$, then the minimal separator which x_i generates will be $\{x_9, x_7, x_6\}$.

Algorithm Minseps-Maxcliques

input : A chordal graph $G = (V, E)$.
output: Set $\mathscr{S}$ of minimal separators of G ;
Set S of minimal separator generators of G ;
Set $\mathscr{K}$ of maximal cliques of G ;
Set K of maximal clique generators of G ;

init: $G_{NUM} \leftarrow G(\emptyset)$; $G_{ELIM} \leftarrow G$;
$\mathscr{S} \leftarrow \emptyset$; $S \leftarrow \emptyset$; $\mathscr{K} \leftarrow \emptyset$; $K \leftarrow \emptyset$;
for $i = n$ **downto** 1 **do**
  Choose a vertex x_i of G_{ELIM} of maximum label ;
  $G_{NUM} \leftarrow G_{NUM} + \{x_i\}$;
  if $i \leq n$ *and* $label(x_i) \neq \lambda$ **then**
    $//x_i$ is a min. sep. generator and x_{i+1} is a max. clique generator
    $S \leftarrow S + \{x_i\}$; $\mathscr{S} \leftarrow \mathscr{S} \cup \{N_{G_{NUM}}(x_i)\}$;
    $K \leftarrow K + \{x_{i+1}\}$; $\mathscr{K} \leftarrow \mathscr{K} + \{(N_{G_{NUM}}(x_i) - \{x_i\}\}$;
  $\lambda \leftarrow label(x_i)$;
  foreach $y \in N_{G_{ELIM}}(x_i)$ **do**
    Use i to increment the label of y ;
  $G_{ELIM} \leftarrow G_{ELIM} - \{x_i\}$;
$K \leftarrow K + \{x_1\}$; $\mathscr{K} \leftarrow \mathscr{K} + \{N_G(x_1)\}$;

The complexity of the above algorithm is the same as for LexBFS or MCS, which is linear ($O(n + m)$) time.

Example 4.1 Figure 4 below gives a step-by-step example using MCS.
Set of minimal separators: $\mathscr{S} = \{\{c\}, \{g, h\}, \{d\}\}$.
Set of minimal separator generators: $S = \{d, i, e\}$.
Set of maximal cliques: $\mathscr{K} = \{\{a, b, c\}, \{c, d, g, h\}, \{g, h, i\}, \{d, e, f\}\}$.
Set of maximal clique generators: $K = \{c, h, i, f\}$.

- When processing vertex x_6, $x_6 = d$ is defined as a minimal separator generator for $\{c\}$ and thus $x_7 = c$ is a maximal clique generator for $\{a, b, c\}$.
- When processing vertex x_3, $x_3 = i$ is defined as minimal separator generator for $\{g, h\}$, so $x_4 = h$ is a maximal clique generator, for $\{c, d, g, h\}$.
- When processing vertex x_2, $x_2 = h$ is defined as minimal separator generator for $\{d\}$, so $x_3 = i$ is a maximal clique generator for $\{g, h, i\}$.

- At the end, $x_1 = f$ is defined a maximal clique generator, for $\{d, e, f\}$.

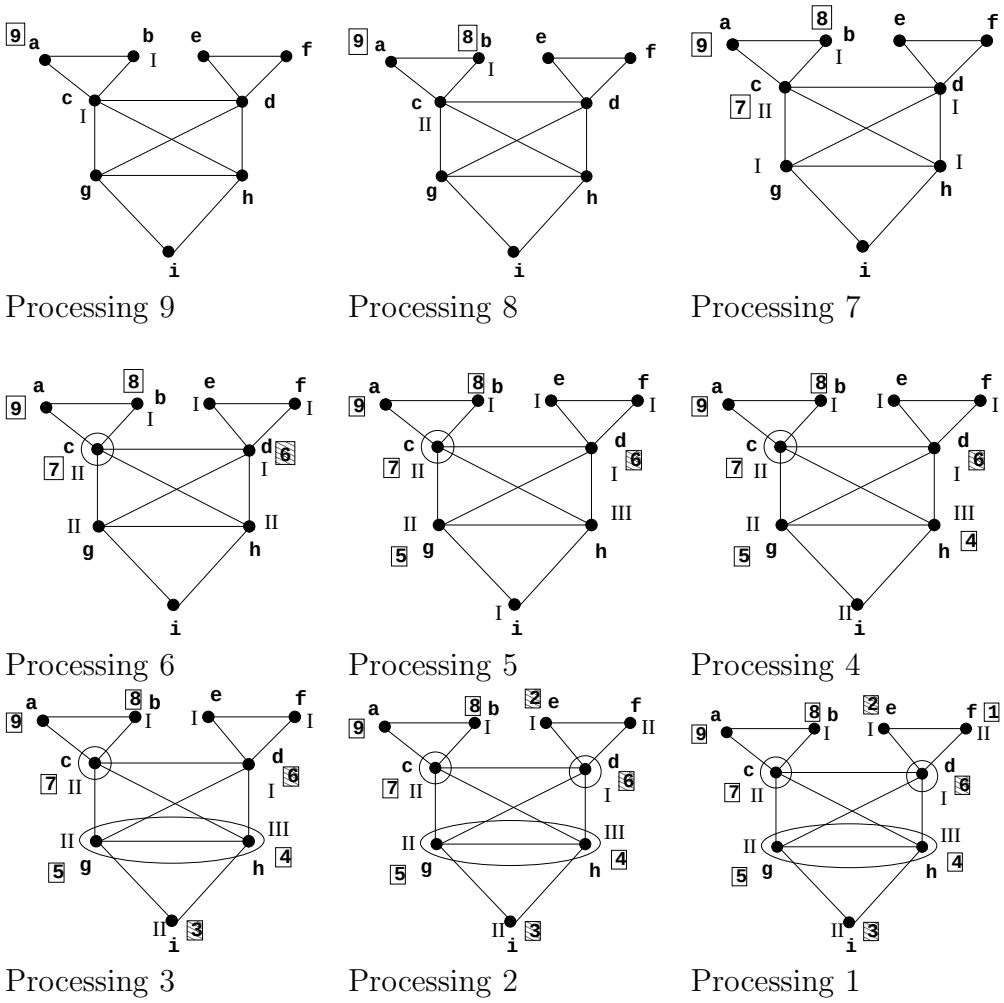

Figure 2: A step by step execution using MCS. The dashed vertex numbers represent the minimal separator generators, and the circled vertex sets are the minimal separators.

4.1 Multiplicity of the generated minimal separators

A minimal separator may be generated several times, depending on the number of full components it defines:

Property 4.2 *Let α be a peo defined by LexBFS or MCS, let S be a minimal separator, let k be the number of full components of S. Then S has $k-1$ generators by α.*

Proof: In the course of a simplicial elimination process on simplicial vertices, before any vertex s of a minimal separator S can be eliminated, all the full components of S (except one) must have been eliminated, else s cannot be simplicial, as s sees the confluence points of each remaining full component. The last vertex eliminated from these full components generates S. When all full components (except one) have been eliminated, S is no longer a minimal separator. Therefore if S has k full components, it is generated exactly $k-1$ times. □

Note that [8] gives an $O(n + mlogn)$ process to find the multiplicity of a minimal separator (with MCS). However, they also prove that with LexBFS, given the minimal separator generator of lowest number which generates S, within the subset of minimal separator generators which define a minimal separator of size $|S|$, all the generators of S are consecutive. In the course of an execution of LexBFS, one can store the labels of the generators in different lists according to the size of the minimal separator they generate. Then one can run through each list, testing for consecutive occurrences of a given minimal separator. This can be done in linear time, as by Theorem 3.7, the sum of the sizes of the minimal separators is less than m.

5 Conclusion

In this paper, we present a simple and optimal process which generates the minimal separators and maximal cliques of a chordal graph in a single pass of either LexBFS or MCS, without requiring the preliminary computation of a peo.

Though both LexBFS and MCS yield an optimal linear-time process for this problem, it is important to note that they define a different set of peos of a chordal graph [4], and exhibit different local behaviors [3]. It may be important to use one or the other, depending on the intended application.

We will end this paper by noting that two recently introduced search algorithms, LexDFS and MNS [6] fail to behave in the same fashion. A counter-example for MNS is given below (The MNS labels are the sets of higher-numbered neighbors, compared by set inclusion): when vertex 2 is numbered, its label is not greater than that of vertex 3; however, $Madj(3) = \{3, 4\}$ is not a maximal clique. .

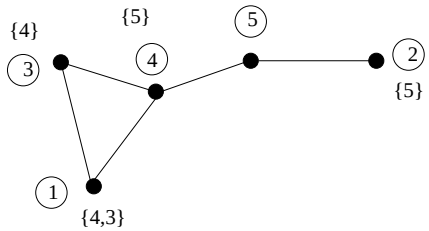

Figure 3: MNS does not generate the maximal cliques.

References

[1] A. Berry, J.-P. Bordat Separability generalizes Dirac's theorem. *Discrete Applied Mathematics*, 84(1-3):43–53, 1998.

[2] A. Berry, J.-P. Bordat Moplex elimination orderings. *Electronic Notes in Discrete Mathematics*, 8:6–9, 2001.

[3] A. Berry, J. R. S. Blair, J.-P. Bordat, and G. Simonet. Graph extremities defined by search algorithms. Research report, LIMOS, RR-07-05, 2007, *to appear in Algorithms*.

[4] A. Berry, R. Krueger, G. Simonet Geneviève Simonet. Maximal label search algorithms to compute perfect and minimal elimination orderings. *SIAM J. Discrete Math.*, 23(1):428–446, 2009.

[5] J. R. S. Blair and B. W. Peyton. An introduction to chordal graphs and clique trees. *Graph Theory and Sparse Matrix Computation*.

[6] D. G. Corneil, R. Krueger A unified view of graph searching. *SIAM J. Discrete Math.*, 22(4):1259–1276, 2008.

[7] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 28:565–570, 1969.

[8] P. Sreenivasa Kumar and C. E. Veni Madhavan. Minimal vertex separators of chordal graphs. *Discrete Applied Mathematics*, 89(1-3):155–168, 1998.

[9] D. J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications,*.

[10] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976.

[11] 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(3):566–579, 1984.