Fast generation of all perfect elimination orderings of a chordal graph.

L.S. Chandran* L. Ibarra† F. Ruskey‡ J. Sawada§

January 30, 2002

Abstract

We develop a constant time transposition "oracle" for the set of perfect elimination orderings of chordal graphs. Using this oracle, we can generate a Gray code of all perfect elimination orderings in constant amortized time using known results about antimatroids. Using clique trees, we show how the initialization of the algorithm can be performed in linear time. We also develop two new characterizations of perfect elimination orderings: one in terms of chordless paths, and the other in terms of minimal vertex separators.

Keywords: Perfect elimination ordering, chordal graph, Gray code, antimatroid, 2-processor scheduling, clique tree.

1 Introduction

Chordal graphs are an important class of graphs. An excellent background on chordal graphs is given by Golumbic in [8]. One of their fundamental characterizations is that a graph is chordal if and only if it has a perfect elimination ordering (PEO). In this paper we consider the problem of efficiently exhaustively listing all of the PEOs of a given input graph as a combinatorial Gray code; we list the PEOs so that successive ones differ only by one or two transpositions of adjacent elements. The key to our algorithm is an efficient way of determining whether any particular adjacent transposition in a given PEO results in another PEO. In order to justify our algorithm we prove new characterizations of chordal graphs and correct an error in the literature. To initialize our algorithm we develop a linear time algorithm for finding a PEO that can be optimally scheduled on two processors.

*Indian Institute of Science, India. Research supported by Infosys Fellowship. email: sunil@csa.iisc.ernet.in

†Simon Fraser University, Canada. Research supported by NSERC. email: libarra@csr.csc.uvic.ca

‡University of Victoria, Canada. Research supported by NSERC. email: fruskey@csr.csc.uvic.ca

§University of Victoria, Canada. Research supported by NSERC. email: jsawada@csr.csc.uvic.ca

```
procedure PEO ( t : integer,  G : graph );
begin
    if t > n then PrintIt(f);
    else for each simplicial vertex v ∈ G do begin
            f(t) := v;
            PEO( t + 1, G − v );
end; end;
```

Figure 1: A naïve algorithm for generating all PEOs.

1.1 Background

Let $G = (V, E)$ be a connected undirected graph where V is the set of vertices and E is the set of edges. A graph G is said to be *chordal* if every cycle of length 4 or more contains a *chord* (an edge between two non-consecutive vertices in the cycle).

We use $N(v)$ to denote the neighborhood of the vertex v i.e., the set of all vertices u adjacent to v. For $A \subseteq V$, we let $G(A)$ denote the subgraph of G induced by A. A complete induced subgraph is called a *clique*. A vertex v is *simplicial* if $G(N(v))$ is a clique. The following result is from Dirac [5]:

THEOREM 1 *Every chordal graph G which is not a clique contains at least two non-adjacent simplicial vertices.*

A bijection $f : \{1, 2, \ldots, n\} \rightarrow V$ with $|V| = n$ is called an *ordering* of G. We let f^{-1} denote the inverse of f and thus $f^{-1}(v)$ denotes the position in the ordering assigned to the vertex v. The set $N_f(v)$ is defined to be the set of all neighbors u of v such that $f^{-1}(u) > f^{-1}(v)$, or in other words, all neighbors of v that have a higher position in the ordering f. Using this notation, an ordering f is called a *perfect elimination ordering* (PEO) if for each $v \in V$, $N_f(v)$ is a clique.

Observe that an arbitrary graph may not have a PEO. The following theorem of Fulkerson and Gross [6] characterizes the graphs which have PEOs.

THEOREM 2 *A graph G is chordal if and only if it has a PEO.*

A single PEO of a chordal graph G can be found in linear time using either the LexBFS algorithm [13] or the maximum cardinality search (MCS) algorithm [16]. However, neither of these algorithms can be used to produce every PEO for a given chordal graph. Using the definition, it is possible to generate all PEOs by successively removing simplicial vertices. Such an algorithm is outlined in Figure 1 where the initial call is $\mathsf{PEO}(1, G)$. The difficulty of this algorithm is in maintaining a list of the simplicial vertices. It does not seem possible to update such a list in time faster than $\Theta(\Delta)$ (where Δ is the maximum degree of any vertex in V) since it is possible to create up to Δ new simplicial vertices when a vertex is removed from the graph.

Another algorithm for generating all PEOs is outlined by Shier in [14]. This algorithm follows the approaches of the Lex-BFS and MCS algorithms by finding the simplicial vertices in reverse order. Implementation details are not given for the algorithm, but again it does not seem possible to update the simplicial vertices in time faster than $O(\Delta)$. Thus, until this paper, it was not known how to generate all PEOs in time faster than $O(\Delta)$ per PEO.

The ultimate goal of an algorithm which produces an exhaustive list of a objects, such as PEOs, is one whose total computation is proportional to the number of objects produced. Such algorithms are said to be CAT since the objects are generated in constant amortized time. When analyzing exhaustive listing algorithms, the correct measure of computation is the total amount of data structure change, not including the time taken to print out the object. This is because typical applications only process the part of the object that has undergone some change. As the two algorithms outlined previously for generating all PEOs of a given chordal graph G do not achieve this goal, we present an alternative approach.

An *antimatroid* is a set system $\mathcal{F}$ that is hereditary and closed under union; i.e., (a) for each non-empty $S \in \mathcal{F}$ there is an $x \in S$ such that $S \setminus \{x\} \in \mathcal{F}$, and (b) for each pair $S, T \in \mathcal{F}$ we have $S \cup T \in \mathcal{F}$. The PEOs of a chordal graph form an antimatroid by taking $\mathcal{F}$ to be the set of all sets of vertices that occur in a prefix of a PEO. The PEOs are the so-called *basic words* of that antimatroid. Further information about antimatroids may be found in the books of Korte, Lovász, and Schrader [9] and Björner and Ziegler [1]. In [9] our antimatroid is referred to as the "vertex shelling of a triangulated graph."

A main result of this paper is the development of a constant time transposition oracle which answers the following question "Is $\tau_j f$ a PEO given that f is a PEO?". By τ_j we denote the transposition $(j\ j{+}1)$. The notation $\tau_j f$ is the result of the action of the permutation τ_j on the indices of f. In other words, the ordering $\tau_j f$ is the ordering f with the adjacent values $f(j)$ and $f(j{+}1)$ transposed. That is, $\tau_j f(j) = f(j{+}1)$ and $\tau_j f(j{+}1) = f(j)$, and for all other $i \in 1, \ldots, n$, $\tau_j f(i) = f(i)$. Since the PEOs of a chordal graph are the basic words of a language antimatroid, we can use this oracle along with an antimatroid result from [12] to generate all PEOs of a chordal graph in constant amortized time using the algorithm $\mathsf{GenLE}(i)$ from [11]. In addition, each successive PEO differs by at most 2 adjacent transpositions. Such algorithms are called *Gray codes* since each successive object differs by a constant amount. The initialization of the algorithm requires an initial PEO obtained by removing *two* simplicial vertices at a time. Using clique trees, we show how this can be done in linear time in Section 5.

Before we describe the constant time transposition oracle in Section 3 and the generation algorithm in Section 4, we first present two new characterizations of perfect elimination orderings: the first is in terms of chordless paths and the second is in terms of minimal vertex separators. Extensive connections between minimal vertex separators and PEOs are shown by Kumar and Madhavan in [10], including a linear time algorithm to identify the minimal vertex separators of a chordal graph. Minimal vertex separators are also used by Simon to prove that the Lex-BFS algorithm is correct [15]. However, in his discussion, a *false* claim is made that the second characterization we present does not hold for every PEO, in particular for some PEOs resulting from the MCS algorithm.

2 New characterizations for PEOs

In this section we develop two new characterizations of perfect elimination orderings. The first characterization is in terms of chordless paths. A chordless path P is a path in which no two non-consecutive vertices are adjacent. For example, the shortest path between any two vertices is chordless.

THEOREM 3 *An ordering f of a chordal graph G is a PEO if and only if every triple of vertices $\{x, y, z\}$ where $xy \in E$, $yz \in E$, and $xz \notin E$ satisfies* $\min(f^{-1}(x), f^{-1}(z)) < f^{-1}(y)$.

PROOF (IF): Suppose there exists a vertex $y \in V$ such that $N_f(y)$ is *not* a clique. Then there exists two non-adjacent nodes x and z in $N_f(y)$ (i.e., $xy \in E$ and $yz \in E$, but $xz \notin E$). Since x and z are in $N_f(y)$, we have $f^{-1}(y) < min(f^{-1}(x), f^{-1}(z))$, a contradiction. Thus for each $y \in V$, $N_f(y)$ is a clique and hence f is a PEO.
(ONLY IF): Consider the triple $\{x, y, z\}$ such that $xy \in E$ and $yz \in E$, but $xz \notin E$. Assume that $f^{-1}(y) < min(f^{-1}(x), f^{-1}(z))$. However since f is a PEO, this means that $N_f(y)$ is a clique implying that $xz \in E$, a contradiction. Thus $min(f^{-1}(x), f^{-1}(z)) < f^{-1}(y)$. □

A sequence $a_1, a_2, \ldots, a_t$ is *unimodal* if there is some value $1 \leq k \leq t$ such that:

$$a_1 \leq a_2 \leq \cdots \leq a_k \geq a_{k+1} \geq \cdots \geq a_t.$$

The following corollary is used in the next section.

COROLLARY 1 *An ordering f is a PEO of a chordal graph G if and only if for every chordless path $P = p_1 \cdots p_t$, the sequence $f^{-1}(p_1), f^{-1}(p_2), \ldots, f^{-1}(p_t)$ is unimodal.*

PROOF: The *if* part of the statement follows immediately from the previous theorem. To prove the *only if* part of the statement, we use induction on the length of the chordless paths. The base case when $t = 3$ follows directly from the previous theorem. Otherwise, consider a chordless path $P = p_1 \cdots p_t$. By induction, we know that the chordless paths $p_1 \cdots p_{t-1}$ and $p_{t-2}p_{t-1}p_t$ are both unimodal. This implies that P must also be unimodal. □

We now give a second characterization of PEOs using minimal separators. A *separator* of G is a subset S of the vertices whose removal separates G into at least two connected components. S is called a *u-v separator* if the vertices u and v are disconnected in $G(V - S)$. A u-v separator is said to be a *minimal u-v separator* if it does not properly contain another u-v separator. S is a *minimal vertex separator* if for some u and v, S is a minimal u-v separator.

THEOREM 4 *An ordering f of a chordal graph G is a PEO if and only if every triple of vertices $\{x, y, z\}$ where $xz \notin E$ and y is in some minimal x-z separator M satisfies $min(f^{-1}(x), f^{-1}(z)) < f^{-1}(y)$.*

PROOF (IF): Suppose there exists a vertex $y \in V$ such that $N_f(y)$ is not a clique. This means that there exist two vertices x and z in $N_f(y)$ that are non-adjacent. Consider a minimal x-z separator M. Clearly $y \in M$ which implies that $min(f^{-1}(x), f^{-1}(z)) < f^{-1}(y)$, contradicting the fact that both x and z are in $N_f(y)$. Thus f is a PEO.
(ONLY IF): Let M be a minimal x-z separator containing y. Let $M' = M - \{y\}$. Since x and z are disconnected in $G(V - M)$ and connected in $G(V - M')$, every path from x to z in $G(V - M')$ contains y. Now consider a shortest (and thus chordless) path from x

to z in $G(V-M')$. Note that such a path is also chordless in G. Thus by Corollary 1, $min(f^{-1}(x), f^{-1}(z)) < f^{-1}(y)$. $\square$

It is interesting to note that Simon (p. 250, [15]) showed that this result was true for all PEOs generated by Lex-BFS. However, he incorrectly claimed that the result did not hold for some other PEOs, in particular, for some generated by the MCS algorithm.

3 A constant time oracle

In this section we show how the question "Is $\tau_j f$ a PEO (given that f is a PEO)?" can be answered in constant time as long as we know the values $|N_f(v)|$ for each vertex $v \in V$. Using this oracle, we can develop a CAT Gray code algorithm for generating all PEOs using the results from [11] and [12]. Details of the algorithm are given in the following section.

Assuming that f is an ordering of a chordal graph G, we let $N_f(j,x)$ denote the set of all $v \in N(x)$ such that $f^{-1}(v) > j$. In other words $N_f(j,x)$ contains all the neighbors of x that are in a position higher than j in the ordering f.

The following lemma is used to prove the main theorem of this section.

LEMMA 1 *If f is a PEO of a chordal graph where $x = f(j)$ and $y = f(j+1)$ are adjacent vertices, then $N_f(j+1,x) \subseteq N_f(j+1,y)$.*

PROOF: Since $N_f(x) = \{y\} \cup N_f(i+1,x)$ is a clique, y is adjacent to every vertex in $N_f(i+1,x)$, which implies $N_f(i+1,x) \subseteq N_f(i+1,y)$. $\square$

THEOREM 5 *Let f be a PEO of a chordal graph G where $x = f(j)$ and $y = f(j+1)$. Then $\tau_j f$ is a PEO of G if and only if x and y are not adjacent or $N_f(j+1,x) = N_f(j+1,y)$.*

PROOF: Since f is a PEO, for every $v \in V$, $N_f(v)$ is a clique. This implies that $f' = \tau_j f$ is a PEO if and only if $N_{f'}(x)$ and $N_{f'}(y)$ are cliques. If x and y are not adjacent, then $N_{f'}(x) = N_f(x)$ and $N_{f'}(y) = N_f(y)$ which are both cliques. If x and y are adjacent, then by Lemma 1 we have $N_f(j+1,x) \subseteq N_f(j+1,y)$. If $N_f(j+1,x) = N_f(j+1,y)$ then $N_{f'}(x) = N_f(y)$ and $N_{f'}(y) = N_{f'}(x) \cup \{x\}$ which are also both cliques. Otherwise if $N_f(j+1,x) \subset N_f(j+1,y)$ then $N_{f'}(y)$ is *not* a clique because x which is in $N_{f'}(y)$ is not adjacent to at least one other vertex in $N_{f'}(y)$. $\square$

The following corollary to this theorem allows us to use simple data structures to determine whether or not the ordering $\tau_j f$ is a PEO in constant time.

COROLLARY 2 *Let f be a PEO of a chordal graph G where $x = f(j)$ and $y = f(j+1)$. Then $\tau_j f$ is a PEO of G if and only if x and y are not adjacent or $|N_f(x)| = |N_f(y)| + 1$.*

PROOF: Suppose x and y are adjacent. Then by Lemma 1, $N_f(j+1,x) = N_f(j+1,y)$ if and only if $|N_f(j+1,x)| = |N_f(j+1,y)|$. Now since $N_f(j+1,x) \cup \{y\} = N_f(x)$ and $N_f(j+1,y) = N_f(y)$, the equality $|N_f(j+1,x)| = |N_f(j+1,y)|$ is equivalent to $|N_f(x)| = |N_f(y)| + 1$. $\square$

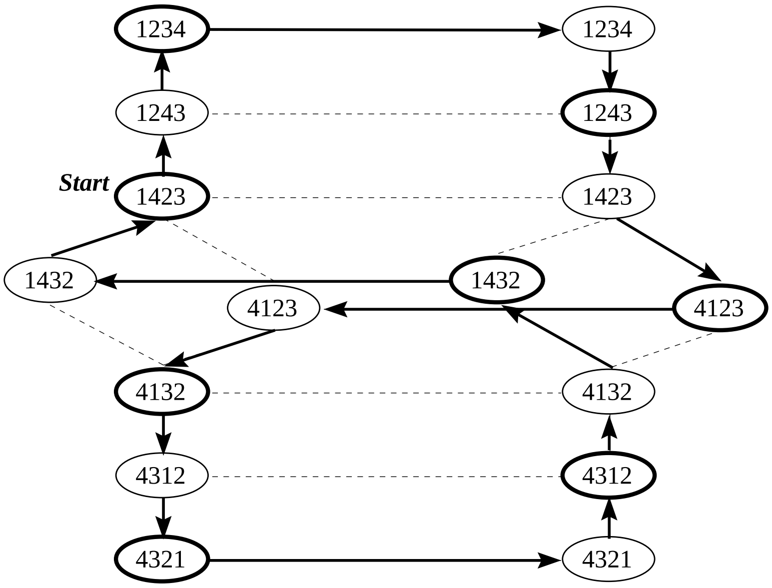

Figure 2: A Hamilton cycle in $H \times e$.

4 The Gray code algorithm

In this section we outline the algorithm for generating all PEOs of a chordal graph G as described in the papers [11] and [12]. The main result of [11] is the development of a Gray code for generating linear extensions of a partially ordered set using the algorithm $\mathsf{GenLE}(i)$. Later, in [12], it is shown that the same algorithm can be used to generate all the basic words of any antimatroid language with changes made only to the initialization and to the oracle which answers correctly when asked whether two adjacent elements can be transposed. Since the PEOs of a chordal graph G form the basic words of an antimatroid language, we can take advantage of the algorithm. Since the oracle (Corollary 2) can be implemented in constant time by maintaining a simple data structure, the generation algorithm runs in constant amortized time.

Let $\mathcal{P}$ denote the set of all PEOs of a chordal graph G. Consider the graph $H = (\mathcal{P}, E')$ where an edge (u, v) is in E' if an only if the PEOs u and v differ by a single adjacent transposition. The *prism* of H, denoted $H \times e$ where $e \in E'$, is the graph which results from taking two copies of H and adding edges between the vertices corresponding to the same PEO. The basic idea behind the Gray code algorithm is to traverse a particular Hamilton cycle in the graph $H \times e$. Such a traversal will visit each PEO exactly twice. However, from Theorem 5.5 from [11], if we print only every second PEO visited in the Hamilton cycle, we obtain every PEO exactly once. As an example, Figure 2 illustrates the graph $H \times e$ obtained from the graph G which is a simple path [1,2,3,4]. Notice that there are two copies of the graph H with edges between the vertices corresponding to the same PEOs. A Hamilton cycle is also illustrated starting from the PEO 1423.

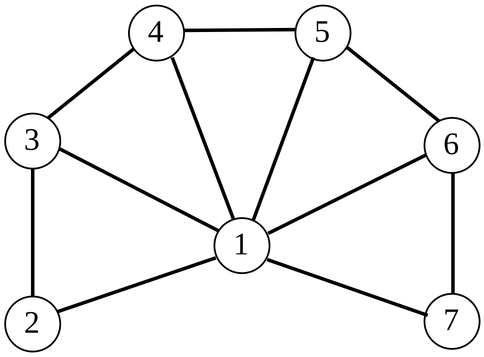

Figure 3: A chordal graph with 7 vertices.

4.1 Initialization

The algorithm assumes that the adjacency matrix of the chordal graph is stored in $G[\ ][\ ]$ and that there are n vertices in the graph. The other data structures required are as follows:

- $peo[12\cdots n]$: the PEO,
- $inv[12\cdots n]$: the inverse of peo,
- $a[i], b[i]$: simplicial vertex pair,
- $h[x]$: the value $N_f(x)$.

To start the algorithm for finding a particular Hamilton cycle, we require an initial PEO that is obtained by removing pairs of simplicial vertices from G. From Theorem 1 we know that every chordal graph (with more than one vertex) has at least two simplicial vertices. The i-th simplicial vertex pairs are stored in $a[i]$ and $b[i]$ respectively. To obtain such a PEO for a general chordal graph, we cannot use linear time algorithms like LexBFS or MCS for finding an initial PEO because such algorithms do not remove pairs of simplicial vertices, but instead remove them one at a time.

Example: Consider the chordal graph in Figure 3. It has two simplicial vertices 2 and 7. The PEO (2,3,4,5,6,7,1) is one that can be computed in linear time using the MCS algorithm. However, initially the vertex 3 is not a simplicial vertex. It does not become simplicial until the vertex 2 is removed from the graph. Hence the initial pair of (2,3) is invalid. Since 2 and 7 are the only simplicial vertices, they must be the first pair to be removed from G. Hence $a[1]$ and $b[1]$ get assigned the values 2,7 or 7,2 respectively. Once these vertices are removed, the updated graph has simplicial vertices 3 and 6 which get assigned in some order to $a[2]$ and $b[2]$. When these vertices are removed we are left with 3 simplicial vertices 4,5,1. Now, any pair of these vertices can be assigned to a[3], b[3]. In this case, since n is odd, there remains a final vertex which does not become part of an $a[i]$, $b[i]$ pair. One such example of a valid initial PEO is (2,7,3,6,4,5,1). □

Because the well known linear time algorithms for finding a PEO can not be used, we we need a new method for finding an initial PEO. In general it takes $O(n^2)$ time to determine if

```
procedure InitPEO ( );
local i, x, y : integer;
begin
    for i := 1 to ⌊n/2⌋ do begin
        (x, y) := GetSimplicialPair();
        a[i] := peo[2i − 1] := x;
        b[i] := peo[2i] := y;
        inv[x] := 2i − 1;   inv[y] := 2i;
    end;
    if Odd(n) then begin
        x := the remaining vertex;
        peo[n] := x;
        inv[x] := n;
    end;
    for each vertex v in G do h[v] := N_f(v);
end;
```

Figure 4: InitPEO()

a vertex is simplicial. Thus, in the worst case, to find the first 2 simplicial vertices will take $O(n^3)$ time. If these vertices are removed from the graph, then the search for the next two simplicial vertices will take time $O((n-2)^3)$. Repeating this process yields a fairly simple, but naïve algorithm that runs in time $O(n^4)$ for finding a valid initial PEO. Because the total number of PEOs produced is $\Omega(2^n)$, the initialization is not a factor in the overall running time. However, it is an interesting question in itself as to how efficiently we can perform this initialization step. In the following section, we describe a more complex approach using clique trees, that removes pairs of simplicial vertices in linear time.

Assuming that the function GetSimplicialPair() returns two simplicial vertices from the chordal graph (and also temporarily removes the vertices from the graph) then pseudocode for this initialization step is shown in Figure 4. As the pairs of simplicial vertices are removed, the appropriate initializations are performed for the data structures $peo[\,]$,$inv[\,]$,$a[\,]$,$b[\,]$. If n is odd, then the last remaining vertex is placed in $peo[n]$. Once the PEO has been completely initialized, we compute the values for $h[\,]$.

4.2 Traversing the Hamilton cycle

Now that we have initialized our data structures we show routine GenPEO(i) in Figure 5 which is a modification of GenLe(i) tailored for PEOs. The only changes are the parameters passed to the subroutines Switch(t) and Move(t) and the test which determines whether two adjacent elements are swappable. The latter test is answered by the routine Swappable(t).

The subroutine Move(t) swaps the elements $peo[t]$ with $peo[t+1]$ updating the arrays $inv[\,]$ and $h[\,]$ as required. If $t \neq 0$ then the subroutine Switch(t) swaps the elements $a[t]$ and $b[t]$ in the peo array and also swaps the values $a[t]$ and $b[t]$ themselves. Again, the arrays $inv[\,]$ and $h[\,]$ must be updated accordingly. If $t = 0$, then the subroutine Switch(t) effectively moves to the duplicate PEO. As each of these two subroutines forces a move to a new PEO in the Hamilton cycle, the function PrintIt() is called to print the PEO. But recall, since every PEO occurs exactly twice, the PrintIt() function only prints out a PEO every second time it is called. Pseudocode for the routines Move(t) and Switch(t) are shown in Figures 6 and 7 respectively.

The subroutine Swappable(t) simply returns whether or not the two adjacent elements $peo[t]$ and $peo[t+1]$ are swappable to give a new PEO. This can be done in constant time using Corollary 2 as long as the values $h[x] = N_f(x)$ are maintained as the elements are swapped. Pseudocode for Swappable(t) is shown in Figure 8.

The initial sequence of calls to generate all PEOs of the input graph G are: InitPEO(); PrintIt(); GenPEO($\lfloor n/2 \rfloor$); Switch($\lfloor n/2 \rfloor$); GenPEO($\lfloor n/2 \rfloor$);.

THEOREM 6 *The algorithm GenPEO(i) for generating all PEOs of a given chordal graph G runs in constant amortized time.*

5 Initialization in linear time

In this section we show that the initialization of the generation algorithm can be carried out in linear time, $O(n+m)$, where n is the number of vertices in the chordal graph and m is the number of edges. This shows that the vertices of a chordal graph can be "scheduled" optimally on two processors by an algorithm that takes linear time. The vertices are constrained so that only a simplicial vertex can be run. An optimal schedule is one of minimal length, length $\lceil n/2 \rceil$ in light of Theorem 1. This scheduling is analogous to, but simpler than, the scheduling of partially ordered sets on two machines; e.g., Coffman and Graham [4].

5.1 Clique trees

A clique K is *maximal* if K is not properly contained in another clique, or equivalently, if no vertex in $V - K$ is adjacent to every vertex in K. A *clique tree* of G is a tree T on the maximal cliques of G such that T has the *clique intersection property*: for any two maximal cliques K, K', the set $K \cap K'$ is contained in every clique on the path from K to K' in T. Buneman [3], Gavril [7], and Walter [17] independently discovered the following characterization of chordal graphs.

THEOREM 7 *A graph G is chordal if and only if G has a clique tree.*

Figure 9 shows a chordal graph G and Figure 10 shows a clique tree of G. Replacing the edge $\{K_y, K_w\}$ with $\{K_x, K_w\}$ gives a different clique tree for G, which shows that a graph's clique tree may not be unique.

A chordal graph has at most n maximal cliques [6], so its clique tree has at most n nodes. There are numerous algorithms that compute the maximal cliques of G and a clique tree T of G. Blair and Peyton [2] describe a simple $O(m+n)$ time algorithm in their primer on chordal graphs and clique trees. It is also known that the sum of all vertices over each maximal clique is in $O(m+n)$ (see, for example pages 98-99 in [8]).

5.2 The algorithm

Let G be a connected chordal graph. We present an algorithm that finds a PEO $f(1), f(2), \ldots, f(n)$ of the vertices of G such that for every odd $1 \leq i \leq n-1$, $f(i)$ and $f(i+1)$ are simplicial vertices of $G(\{f(i), f(i+1), \ldots, f(n)\})$. If this induced subgraph is a clique, then every vertex is simplicial and adjacent. Otherwise, the vertices v_i, v_{i+1} are selected so that they are non-adjacent (two such vertices exist by Theorem 1). The algorithm maintains a clique tree of the graph, which it uses to find and update the simplicial vertices quickly.

1. Compute a clique tree T of G. Compute a linked list $leaves(T)$ containing the leaves of T.

2. For each vertex v, compute the number $num(v)$ of maximal cliques of G that contain v, which may be done by scanning the vertices of all of the maximal cliques. For every maximal clique K, compute a linked list $simplicial(K)$ of the simplicial vertices contained in K, which are those vertices $v \in K$ such that $num(v) = 1$.

3. Choose any two maximal cliques $K_1, K_2 \in leaves(T)$. Choose any two simplicial vertices $v_1 \in K_1, v_2 \in K_2$. Print and delete v_1, v_2. For each $i = 1, 2$ in sequence, if $simplicial(K_i)$ is empty, then let K_i' be K_i's neighbor in T and do the following two steps.

   (a) For every vertex $u \in K_i$, decrement $num(u)$ by 1. and if the new value of $num(u)$ is 1, then add u to $simplicial(K_i')$.

   (b) Delete K_i from T and $leaves(T)$. If K_i' has degree 1 in T, then add K_i' to $leaves(T)$.

4. If T contains least 2 nodes, go to step 3. Otherwise, print the remaining vertices.

Correctness

A vertex is simplicial if and only if it is contained in exactly one maximal clique; see [2] for the history of this observation. Then in any clique tree T, any leaf is a maximal clique K that contains at least one simplicial vertex; otherwise, the clique intersection property implies that every vertex of K is contained in K's neighbor K' in T, which means $K \subset K'$, a contradiction. Therefore, the first two vertices v_1, v_2 chosen by the algorithm are simplicial. Moreover, v_1, v_2 are non-adjacent; otherwise, clique $\{v_1, v_2\}$ is contained in some maximal clique K_3, contradicting the fact that v_1, v_2 are simplicial. Observe that $G[V - \{v_1, v_2\}]$ is chordal (because every induced subgraph of a chordal graph is chordal) and connected (because deleting any simplicial vertex from a connected graph yields a connected graph). Thus, it suffices to show that after deleting v_1, v_2, the algorithm correctly updates the data structure.

After a maximal clique K is placed in $leaves(T)$, its simplicial vertices are deleted one by one. After K's last simplicial vertex is deleted, K's remaining vertices are non-simplicial and therefore contained in K's neighbor K' in T. Then K is not a maximal clique of the current G and K must be deleted from T. For every remaining vertex u of K, the algorithm decrements $num(u)$ and adds u to $simplicial(K')$ if u is simplicial in the current G. Thus, the algorithm correctly updates the data structure.

Complexity

Since steps 1 and 2 run in $O(m+n)$ time, consider step 3. When a simplicial vertex v is deleted, charge the operation to the maximal clique that contains v. When a non-simplicial vertex u has $num(u)$ decremented, charge the operation to the maximal clique that contains v and that is about to be deleted from T. Then each maximal clique K is charged at most $|K|$. Thus, step 3 runs in time $O(\sum |K|) = O(m+n)$.

6 Summary

The main result in this paper is the discovery of a constant time oracle that answers the question of whether two adjacent elements of a PEO can be swapped to obtain a new PEO. This leads to the first algorithm for generating all PEOs of a chordal graph G in constant amortized time. As an interesting subproblem, we show how the initialization of the algorithm can be performed in linear time using clique trees.

In addition, we develop two new characterizations of PEOs: one in terms of chordless paths, the other in terms of minimal vertex separators. The latter result disproves a claim made by Simon in [15]. The algorithm has been implemented (using the slower initialization) and may be obtained from the last two authors. It would be interesting to extend our results to the basic words of other natural antimatroids.

References

[1] A. Björner and G.M. Ziegler, "Introduction to Greedoids", in *Matroid Applications* (ed. N. White), Cambridge University Press, 1992.

[2] J.R.S. Blair and B. Peyton, An introduction to chordal graphs and clique trees, Graph Theory and Sparse Matrix Computation, Springer-Verlag, 56 (1993) 1-29.

[3] P. Buneman, A characterization of rigid circuit graphs, Discrete Mathematics, 9 (1974) 205-212.

[4] E. Coffman and R.L. Graham, Optimal scheduling for two-processor systems, Acta Informatica, 1 (1972) 200–213.

[5] G.A. Dirac, On rigid circuit graphs, Abh. Math. Sem. Univ. Hamburg, 24 (1961) 71-76.

[6] D.R. Fulkerson and O.A. Gross, Incidence matrices and interval graphs, Pacific Journal of Mathematics, 15 (1965) 835-855.

[7] F. Gavril, The intersection graphs of subtrees in trees are exactly the chordal graphs, Journal of Combinatorial Theory B, 16 (1974) 47-56.

[8] M.C. Golumbic, *Algorithmic graph theory and perfect graphs*, Academic Press, 1980.

[9] B. Korte, L. Lovász, R. Schrader, *Greedoids*, Springer-Verlag, 1991.

[10] P.S. Kumar and C.E. Veni Madhavan, Minimal vertex separators of chordal graphs, Discrete Applied Mathematics, 89 (1998) 155-168.

[11] G. Pruesse and F. Ruskey, Generating linear extensions fast, SIAM Journal on Computing, Vol. 23 No. 2 (1994) 373-386.

[12] G. Pruesse and F. Ruskey, Gray codes for antimatroids, Order, 10 (1993) 239-252.

[13] D. Rose, R. Tarjan, and G. Lueker, Algorithmic aspects of vertex elimination on graphs, SIAM Journal on Computing, 5 (1975) 266-283.

[14] D.R. Shier, Some aspects of perfect elimination orderings in chordal graphs, Discrete Applied Mathematics, 7 (1984) 325-331.

[15] K. Simon, A note on lexicographic breadth first search for chordal graphs, Information Processing Letters, 54 (1995) 249-251.

[16] R. Tarjan and M. Yannakakis, Simple linear-time algorithm to test chordality of graphs, test acyclicity of hypergraphs, and selectivity reduce acyclic hypergraphs, SIAM Journal on Computing, 13 (1984) 566-579.

[17] J.R. Walter, Representations of chordal graphs as subtrees of a tree, Journal of Graph Theory, 2 (1978) 265-267.

```
procedure GenPEO ( i: integer);
local j, typical, mrb, mra, mla : integer;
begin
    if i = 0 then return;
    GenPEO( i − 1 );
    mrb := typical := 0;
    while Swappable( inv[b[i]] ) do begin
        mrb := mrb + 1;
        Move( inv[b[i]] );  GenPEO( i − 1 );
        mra := 0;
        if peo[inv[a[i]] + 1] ≠ b[i] and Swappable( inv[a[i]] ) then begin
            typical := 1;
            do
                mra := mra + 1;
                Move( inv[a[i]] );  GenPEO( i − 1 );
            while peo[inv[a[i]] + 1] ≠ b[i] and Swappable( inv[a[i]] );
        end;
        if typical then begin
            Switch( i − 1 );  GenPEO( i − 1 );
            if Odd( mrb ) then mla := mra − 1;
            else mla := mra + 1;
            for j := 1 to mla do begin
                Move( inv[a[i]] − 1 );  GenPEO( i − 1 );
    end; end; end;
    if typical and Odd( mrb ) then Move( inv[a[i]] − 1 );
    else Switch( i − 1 );
    GenPEO( i − 1 );
    for j := 1 to mrb do begin
        Move( inv[b[i]] − 1 );  GenPEO( i − 1 );
end; end;
```

Figure 5: GenPEO(i)

```
procedure Move (t: integer );
local x, y : integer;
begin
        x := peo[t];    y := peo[t + 1];
        swap( peo[t], peo[t + 1] );
        swap( inv[x], inv[y] );
        if G[x][y] = 1 then begin
                h[y] := h[y] + 1;    h[x] := h[x] − 1;
        end;
        PrintIt();
end;
```

Figure 6: Move(t)

```
procedure Switch (t: integer );
local x, y : integer;
begin
        if t ≠ 0 then begin
                x := a[t];    y := b[t];
                swap( peo[inv[x]], peo[inv[y]] );
                swap( inv[x], inv[y] );
                swap( a[t], b[t] );
                if G[x][y] = 1 then begin
                        h[y] := h[y] + 1;    h[x] := h[x] − 1;
                end;
        end;
        PrintIt();
end;
```

Figure 7: Switch(t)

```
function Swappable (t: integer ) returns boolean;
local x, y : integer;
begin
        x := peo[t];    y := peo[t + 1];
        if t = n then return(FALSE);
        return( G[x][y] = 0 or h[x] = h[y] + 1);
end;
```

Figure 8: Swappable(t)

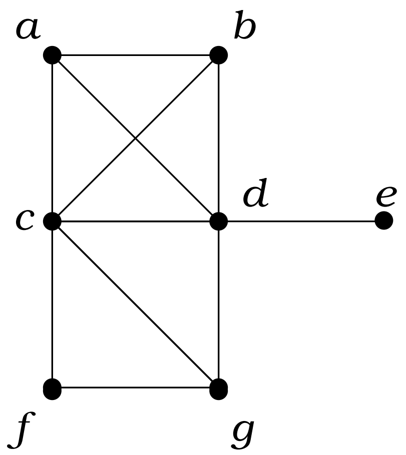

Figure 9: A chordal graph G.

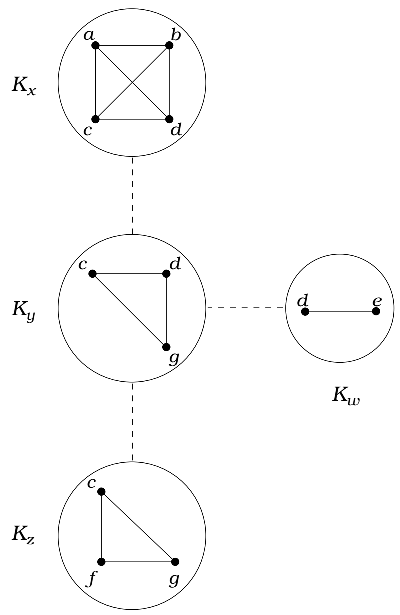

Figure 10: A clique tree T of the graph in Figure 9, with T's edges shown by dashed lines.