数理解析研究所講究録
第1644巻 2009年 86-94

Enumeration of Perfect Sequences of Chordal Graph

Yasuko Matsui[1], Ryuhei Uehara[2], and Takeaki Uno[3]

[1] Department of Mathematical Sciences, School of Science, Tokai University, Kitakaname 1117, Hiratsuka, Kanagawa 259-1292, Japan. yasuko@ss.u-tokai.ac.jp
[2] School of Information Science, JAIST, Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan. uehara@jaist.ac.jp
[3] National Institute of Informatics, Hitotsubashi 2-1-2, Chiyoda-ku, Tokyo 101-8430, Japan. uno@nii.jp

Abstract. A graph is chordal if and only if it has no chordless cycle of length more than three. The set of maximal cliques in a chordal graph admits special tree structures called clique trees. A perfect sequence is a sequence of maximal cliques obtained by using the reverse order of repeatedly removing the leaves of a clique tree. This paper addresses the problem of enumerating all the perfect sequences. Although this problem has statistical applications, no efficient algorithm has been proposed. There are two difficulties with developing this type of algorithms. First, a chordal graph does not generally have a unique clique tree. Second, a perfect sequence can normally be generated by two or more distinct clique trees. Thus it is hard using a straightforward way to generate perfect sequences from each possible clique tree. In this paper, we propose a method to enumerate perfect sequences without constructing clique trees. As a result, we have developed the first polynomial delay algorithm for dealing with this problem. In particular, the time complexity of the algorithm on average is $O(1)$ for each perfect sequence.
Keywords: chordal graph, clique tree, enumeration, perfect sequence.

1 Introduction

A graph is said to be chordal if every cycle of length at least 4 has a chord. Chordal graphs have been investigated for a long time for many areas. From the viewpoint of graph theory, this class has a simple characterization; a graph is chordal if and only if it is an intersection graph of the subtrees of a tree. That is, there is a set of subtrees T_v of a tree $\mathcal{T}$ that correspond to the vertices v of a chordal graph G such that u and v are adjacent in G if and only if the corresponding subtrees T_u and T_v have non-empty intersection. From an algorithmic point of view, a chordal graph is characterized by a simple vertex ordering called a perfect elimination ordering (PEO). Its geometrical property of rigidity plays an important role in many practical areas, and the property of its adjacency matrix is useful in matrix manipulations. Since the class appears in different contexts in so many areas they also are called "rigid circuit graphs" or "triangulated graphs" (see e.g., [13, 5]).

Recently, the probabilistic relationships between random variables are represented by a graph called "graphical model" in the science of statistics. On a graphical model, the random variables are represented by the vertices and the conditional dependencies are represented by the edges. In particular, if a graphical model is chordal, we can easily compute its maximum likelihood estimator. In the computation, a chordal graph is decomposed into its subgraphs by removing a separator which induces a clique. Therefore, chordal graphs are also called "decomposable graphs" or "decomposable models" in this area (see, e.g., [9]). Even if a given graphical model is not decomposable, we sometimes add edges to make it decomposable to obtain a good approximation value of the maximum likelihood estimator of the graphical model (see, e.g., [4]). This corresponds to the notion of "chordal completion" in the area of graph algorithms [8]. Hence chordal graphs play an important role in graphical modeling in statistics.

Perfect sequences are the sequences of maximal cliques in a given chordal graph that satisfy certain properties. The notion arises from the decomposable models, and all perfect sequences are required and

must not have repetitions (see, e.g., [7]) From the viewpoint of graph theory, perfect sequences can be seen in the following way. As mentioned, a chordal graph G can be represented by the intersection graph of the subtrees of a tree $\mathcal{T}$. That is, each vertex v of $G = (V, E)$ corresponds to a subtree T_v of $\mathcal{T}$, and $\{u, v\} \in E$ if and only if T_v and T_u intersect. We can make each node c_i of tree $\mathcal{T}$ correspond to a maximal clique C_i of G; C_i consists of all the vertices v in G such that T_v contains the node c_i. Therefore, the tree $\mathcal{T}$ is called a clique tree of G. From the clique tree $\mathcal{T}$, we make an ordering π over the set of maximal cliques $\{C_1, C_2, \ldots, C_k\}$ of G such that $C_{\pi(i)}$ is a leaf of tree $\mathcal{T}_i$ which is a subgraph of $\mathcal{T}$ induced by $C_{\pi(1)}$, $C_{\pi(2)}$, $\ldots$, $C_{\pi(i)}$ for each i. Intuitively, we can construct such a sequence from $\mathcal{T}$ by repeatedly pruning leaves and put them on the top of the sequence until $\mathcal{T}$ is empty. Then the sequence of maximal cliques in a chordal graph is called a perfect sequence, and it is known that a graph is chordal if and only if it has a perfect sequence.

In 2006, Hara and Takemura proposed a sampling algorithm for the perfect sequences of a given chordal graph [6] that used the Lauritzen's method [9]. However their algorithm does not generate each perfect sequence uniformly at random, and to our knowledge, no enumeration algorithm of perfect sequences exists. There are two major reasons for the difficulty in enumerating perfect sequences. First, the clique tree is not generally unique for a chordal graph. That is, a chordal graph has many distinct (non-isomorphic) clique trees in general. For a clique tree, we can define a set of perfect sequences consistent to the clique tree. Then, secondly, the sets of perfect sequences consistent with the distinct clique trees are not disjoint. That is, we can obtain one perfect sequence from possibly many distinct clique trees. Therefore, a straightforward algorithm based on a simple idea (generate all clique trees, and generate all perfect sequences for each clique tree) cannot avoid redundancy.

In this paper, we propose an algorithm enumerating all the perfect sequences of a chordal graph. The algorithm enumerates all the perfect sequences on an average of $O(1)$ time per sequence. In order to avoid redundancy, our algorithm makes a weighted intersection graph of maximal cliques first instead of explicitly constructing the clique trees. The intersection graph is uniquely constructed, and each maximum weighted spanning tree of the intersection graph gives a clique tree of a chordal graph. Then the algorithm generates each perfect sequence from the union of maximum weighted spanning trees without any repetitions. The algorithm is based on a new idea that characterizes the union of maximum weighted spanning trees, and that also gives us insight into the properties of the set of clique trees of a chordal graph.

We note that the set of perfect sequences is strongly related to the set of PEOs. The PEO is a standard characterization of a chordal graph in the area of graph algorithms. Any PEO can be obtained by repeatedly removing a simplicial vertex, and any sequence of removals of simplicial vertices is a PEO. This property admits us to enumerate all PEOs by recursively removing simplicial vertices. The enumeration of all PEOs is investigated by Chandran et al. [3]. Intuitively, any perfect sequence can be obtained by removing a set of "equivalent" simplicial vertices together repeatedly. However, each removal has to follow (a kind of) lexicographical ordering. A removal of a set of simplicial vertices may produce a new set of simplicial vertices or change another set of simplicial vertices, and the new set cannot be chosen before the old sets. Thus the family of sets of equivalent simplicial vertices has to follow a partial ordering defined by the appearance in the graph to obtain a perfect sequence. This fact also implies that while some PEOs can be obtained from a perfect sequence easily, but some PEOs cannot be obtained from any perfect sequence straightforwardly by the constraint of the partial ordering. Therefore, the correspondence between the set of perfect sequences and the set of PEOs is not straightforward and hence we have to analyze some special cases. Indeed, using this approach, we can obtain another enumeration algorithm of all perfect sequences from the enumeration algorithm of all PEOs. However this approach does not allow us to enumerate efficiently; the algorithm takes $O(|V| + |E|)$ time for each perfect sequence. This

is the reason why we take completely different approach based on a maximum weighted spanning tree of a weighted clique graph, which admits us to improve the time to $O(1)$ on average.

2 Preliminaries

The *neighborhood* of a vertex v in a graph $G = (V, E)$ is the set $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$, and the *degree* of a vertex v is $|N_G(v)|$ and is denoted by $\deg_G(v)$. For a vertex subset U of V we denote by $N_G(U)$ the set $\{v \in V \mid v \in N_G(u) \text{ for some } u \in U\}$. If no confusion arises we will omit the subscript G. Given a graph $G = (V, E)$ and a subset $U \subseteq V$, the *subgraph of G induced by U* is the graph (U, F), where $F = \{\{u, v\} \in E \mid u, v \in U\}$, and denoted by $G[U]$. A vertex set I is an *independent set* of G if $G[I]$ contains no edge, and a vertex set C is a *clique* in G if any pair of vertices in C is connected by an edge in G. An edge e in a connected graph $G = (V, E)$ is called a *bridge* if its removal partitions G into two connected components.

For a given graph $G = (V, E)$, consider a sequence $(v_0, v_1, \ldots, v_\ell)$ of vertices in V such that $\{v_{j-1}, v_j\} \in E$ for each $0 < j \leq \ell$. Such a sequence is a *path* if the vertices $v_0, \ldots, v_\ell$ are all distinct, and it is a *cycle* if the vertices $v_0, \ldots, v_{\ell-1}$ are distinct and $v_0 = v_\ell$. The *length* of such a path and a cycle is the number ℓ. An edge that joins the two vertices of a cycle, but is not itself an edge of the cycle, is a *chord* of the cycle. A graph is *chordal* if each cycle of length at least 4 has a chord.

Given a graph $G = (V, E)$, a vertex $v \in V$ is *simplicial* in G if $N(v)$ is a clique in G. An ordering $v_1, \ldots, v_n$ of the vertices of V is a *perfect elimination ordering* of G if the vertex v_i is simplicial in $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ for all $i = 1, \ldots, n$. It is known that a graph is chordal if and only if it has a perfect elimination ordering [2, Section 1.2]. Given a chordal graph, a perfect elimination ordering of the graph can be found in linear time [12, 14].

For a chordal graph $G = (V, E)$, we can associate a tree $\mathcal{T}$, called a *clique tree* of G, as satisfying the following three properties. (A) The nodes[4] of $\mathcal{T}$ are the maximal cliques of G. (B) Two nodes of $\mathcal{T}$ are adjacent only if the intersection of the corresponding maximal cliques is non-empty. (C) For every vertex v of G, the subgraph $\mathcal{T}_v$ of $\mathcal{T}$ induced by the maximal cliques containing v is connected. (Here, condition (A) is sometimes weakened as each node is not necessarily maximal.) It is well known that a graph is chordal if and only if it has a clique tree, and in such a case a clique tree can be constructed in linear time. On the tree, each vertex v in V corresponds to a subtree $\mathcal{T}_v$ of $\mathcal{T}$. That is, $\mathcal{T}_v$ consists of maximal cliques that contain v. Then, the graph G is an intersection graph of subtrees $\mathcal{T}_v$ of a tree $\mathcal{T}$. Some of these details are explained in books [2, 13].

For a given chordal graph $G = (V, E)$, we denote the set of all maximal cliques of G by $\mathcal{C}(G)$. (It is known that $|\mathcal{C}(G)| \leq |V|$.) Let $k = |\mathcal{C}(G)|$, $\mathcal{C}(G) = \{C_1, C_2, \ldots, C_k\}$, and π be a permutation of k elements. Then, the ordering $C_{\pi(1)}, C_{\pi(2)}, \ldots, C_{\pi(k)}$ on $\mathcal{C}(G)$ is said to be a *perfect sequence* if there is a clique tree $\mathcal{T}$ such that each $C_{\pi(i)}$ is a leaf of the subtree $\mathcal{T}[\{C_{\pi(1)}, C_{\pi(2)}, \ldots, C_{\pi(i)}\}]$ which is the subgraph of $\mathcal{T}$ induced by $\{C_{\pi(1)}, C_{\pi(2)}, \ldots, C_{\pi(i)}\}$, for each i with $1 \leq i \leq k$. Intuitively, we have two explanations for this. One is that we can prune all the leaves off a clique tree in the reverse order $C_{\pi(k)}, \ldots, C_{\pi(2)}, C_{\pi(1)}$ of any perfect sequence. On the other hand, according to the perfect sequence $C_{\pi(1)}, C_{\pi(2)}, \ldots, C_{\pi(k)}$, we can construct a clique tree $\mathcal{T}$ by repeatedly attaching $C_{\pi(i)}$ as a leaf.

We here note that, in general, a clique tree for a chordal graph G is not uniquely determined up to isomorphism. For example, for the chordal graph $G = (V, E)$ in Fig. 1(a), there are four distinct clique trees given in Fig. 1(b). Moreover, two or more distinct clique trees of the same chordal graph G can generate the same perfect sequence. For example, for the chordal graph $G = (V, E)$ in Fig. 1(a), a perfect sequence $C_2C_1C_3C_4C_5$ can be generated from two trees, T_1 and T_2, as depicted in Fig. 1(b)(c).

[4] In this paper, "vertex" is in a chordal graph G, and "node" corresponds to a clique in G to distinguish them.

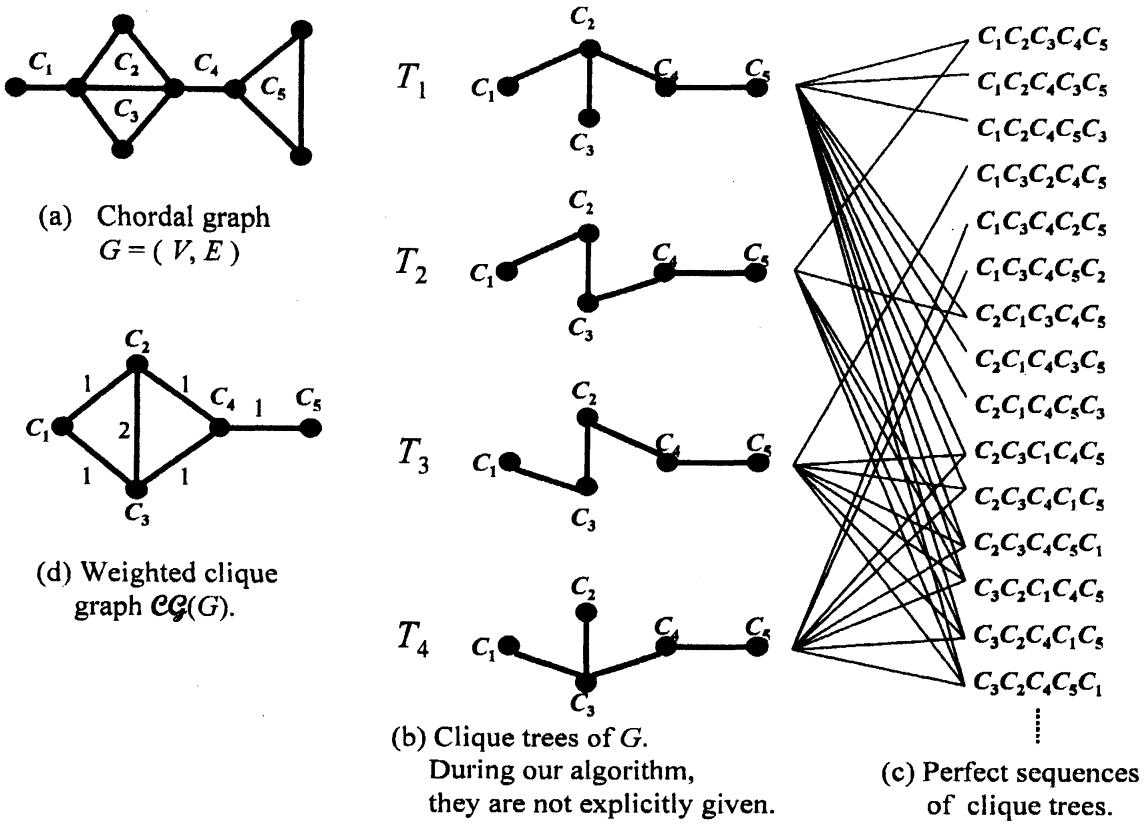

Fig. 1. (a) Chordal graph G, (b) its clique trees, (c) corresponding perfect sequences, and (d) its weighted clique graph $\mathcal{CG}(G) = (\mathcal{C}(G), \mathcal{E})$

For a chordal graph $G = (V, E)$ and the set $\mathcal{C}(G)$ of all maximal cliques, we define the *weighted clique graph* $\mathcal{CG}(G) = (\mathcal{C}(G), \mathcal{E})$ with a weight function $w : \mathcal{E} \to \mathbb{Z}$ as follows. For two maximal cliques C_1 and C_2 in $\mathcal{C}(G)$, $\mathcal{E}$ contains the edge $\{C_1, C_2\}$ if and only if $C_1 \cap C_2 \neq \emptyset$. For each edge $\{C_1, C_2\}$ in $\mathcal{E}$, $w(\{C_1, C_2\})$ is defined by $|C_1 \cap C_2|$ (therefore every edge in $\mathcal{E}$ has a positive integer weight less than $|V|$). The $\mathcal{CG}(G) = (\mathcal{C}(G), \mathcal{E})$ of a chordal graph $G = (V, E)$ in Fig. 1(a) is given in Fig. 1(d). The weights of the edges are all 1 except for $\{C_2, C_3\}$ which has a weight 2.

We recall that each edge in a clique tree $\mathcal{T}$ of G corresponds to a nonempty intersection of two maximal cliques. Thus, $\mathcal{T}$ is a spanning tree of $\mathcal{CG}(G)$. However, some spanning trees of $\mathcal{CG}(G)$ may not be clique trees of G. The characterization of a clique tree is given as follows.

Lemma 1 (e.g., [1, 11]). *Let $G = (V, E)$ be a chordal graph and $\mathcal{CG}(G) = (\mathcal{C}(G), \mathcal{E})$ be the weighted clique graph with a weight function w. A spanning tree $\mathcal{T}$ of $\mathcal{CG}(G)$ is a clique tree of G if and only if it has the maximum weight.*

For the $\mathcal{CG}(G) = (\mathcal{C}(G), \mathcal{E})$ in Fig. 1(d) of the chordal graph $G = (V, E)$ in Fig. 1(a), the only spanning trees that contain the edge $\{C_2, C_3\}$ are clique trees of G. We note that any chordal graph of n vertices contains n maximal cliques at the most. Therefore, $\mathcal{CG}(G)$ contains $O(n)$ nodes. On the other hand, although a star S_n of n vertices contains n vertices, $n-1$ edges, and $n-1$ maximal cliques, the clique graph $\mathcal{CG}(S_n)$ is a complete graph K_{n-1} with $n-1$ nodes that contains $\binom{n-1}{2} = O(n^2)$ edges. Therefore,

we only have a trivial upper bound $O(|V|^2)$ for the number of edges in the clique graph $\mathcal{CG}(G)$ of a chordal graph $G = (V, E)$, even if $|E| = O(|V|)$.

Hereafter, we assume that the input graph G is connected without loss of generality; in case G is not connected, we allow the use of h weight-zero edges to join the h connected components. The modification of the algorithms in this paper is straightforward, and omitted.

We show here a technical lemma for a clique tree that will be referred to later:

Lemma 2. *Suppose $\mathcal{T}$ is a clique tree of a chordal graph $G = (V, E)$ that consists of at least two maximal cliques (in the other words, G is not a complete graph). Let C be a leaf in $\mathcal{T}$ and C' be the unique neighbor of C. Then for each vertex v in C, v is simplicial in G if and only if v is in $C \setminus C'$.* □

3 Enumeration Algorithm

The idea for enumerating perfect sequences is simple. We construct a graph representing the adjacency of maximal cliques, and recursively remove the maximal cliques that can be leaves of a clique tree. Since the clique tree is a spanning tree of the graph and the removed maximal clique is a leaf of the clique tree, after removing the maximal cliques, we still have a clique (sub)tree that is a spanning tree of the resultant graph. Since any tree has at least two leaves, we always have at least two maximal cliques that correspond to the leaves of the spanning tree. Therefore, we invariably get a perfect sequence by repeating the removal process. During the algorithm, the spanning tree is not explicitly given, and we have to deal with all the potential spanning trees that can generate perfect sequences. The following outlines a description of the algorithm.

Algorithm 1: Outline of Enumeration

Input : Chordal graph $G = (V, E)$;
Output: All perfect sequences of G;
1 construct weighted clique graph $\mathcal{CG}(G)$;
2 compute arbitrary maximum weighted spanning tree $\mathcal{T}^*$ of $\mathcal{CG}(G)$;
3 construct graph $\mathcal{CG}(G)^*$ composed of edges that can be included in clique trees from $\mathcal{CG}(G)$ and $\mathcal{T}^*$;
4 enumerate all sequences of maximal cliques obtained by repeatedly removing maximal cliques that can be leaves of some clique trees.

The graph $\mathcal{CG}(G)^*$ is a subgraph of $\mathcal{CG}(G)$ that excludes unnecessary edges described later.

To efficiently find the maximal cliques that can be leaves, we first compute any maximum weighted spanning tree $\mathcal{T}^*$. Then, we use a (unweighted) graph $\mathcal{CG}(G)^*$ obtained from $\mathcal{CG}(G)$ and $\mathcal{T}^*$. We say an edge in $\mathcal{CG}(G)$ is *unnecessary* if it cannot be included in any maximum weighted spanning tree of $\mathcal{CG}(G)$. On the other hand, an edge is *indispensable* if it appears in every maximum weighted spanning tree of $\mathcal{CG}(G)$. The other edges are called *dispensable*, which means they appear in some (but not all) maximum weighted spanning trees. Let e be an edge not in $\mathcal{T}^*$. Since $\mathcal{T}^*$ is a spanning tree of $\mathcal{CG}(G)$, the addition of e to $\mathcal{T}^*$ produces a unique cycle C_e which consists of e and the other edges in $\mathcal{T}^*$. We call C_e an *elementary cycle of* e (the terminology comes from the matroid theory). The unnecessary, dispensable, and indispensable edges are characterized by the following lemmas.

Lemma 3. *For an edge e not in $\mathcal{T}^*$, $w(e) \leq w(e')$ holds for any $e' \in C_e \setminus \{e\}$. Moreover, e is unnecessary if and only if $w(e) < w(e')$ holds for any $e' \in C_e \setminus \{e\}$. On the other hand, e is dispensable if and only if $w(e) = w(e')$ holds for some $e' \in C_e \setminus \{e\}$.* □

Lemma 4. *An edge e in $\mathcal{T}^*$ is an indispensable edge if $w(e) > w(e')$ for all edges e' such that e' is not on $\mathcal{T}^*$ and $C_{e'}$ contains e.*

Proof is analogous to proof of Lemma 3, and hence omitted.

We note that any bridge is indispensable by Lemma 4; there is no edge e' not in $\mathcal{T}^*$ such that $C_{e'}$ contains e and $w(e') \geq w(e)$.

We denote the sets of unnecessary, indispensable, and dispensable edges by $\mathcal{E}_u$, $\mathcal{E}_i$, and $\mathcal{E}_d$, respectively. They partition the edge set $\mathcal{E}$ of $\mathcal{CG}(G)$ into three disjoint sets. The sets can be computed by the following algorithm in $O(|\mathcal{C}(G)|^3) = O(|V|^3)$ time. We note that by using a dynamic programming technique starting from the bottom of the tree, the run time can be reduced to $O(|\mathcal{C}(G)|^2)$, which is omitted here since it is too complex and tedious.

Algorithm 2: Search for Unnecessary, Indispensable, and Dispensable Edges

Input : The weighted clique graph $\mathcal{CG}(G) = (\mathcal{C}(G), \mathcal{E})$ and an arbitrary maximum weighted spanning tree $\mathcal{T}^*$ of $\mathcal{CG}(G)$;

Output: Sets $\mathcal{E}_u$, $\mathcal{E}_i$, and $\mathcal{E}_d$ of the unnecessary, indispensable, and dispensable edges;

1 set $\mathcal{E}_u := \emptyset; \mathcal{E}_d := \emptyset; \mathcal{E}_i := \emptyset$;
2 e not in $\mathcal{T}^*$ $w(e) < w(e')$ for all $e' \in C_e$ $\mathcal{E}_u := \mathcal{E}_u \cup \{e\}$;
3 $\mathcal{E}_d := \mathcal{E}_d \cup \{e\}$;
4 $e' \in C_e$ satisfying $w(e) = w(e')$ $\mathcal{E}_d := \mathcal{E}_d \cup \{e'\}$;
5 $\mathcal{E}_i := E \setminus (\mathcal{E}_u \cup \mathcal{E}_d)$;
6 $(\mathcal{E}_i, \mathcal{E}_u, \mathcal{E}_d)$;

We now define an unweighted graph $\mathcal{CG}(G)^*$ by $(\mathcal{C}(G), \mathcal{E}_i \cup \mathcal{E}_d)$.

Observation 1 *Any spanning tree of $\mathcal{CG}(G)^*$ that contains all the edges in $\mathcal{E}_i$ gives a maximum weighted spanning tree of $\mathcal{CG}(G)$.*

Now, we have characterized clique trees by using spanning trees in $\mathcal{CG}(G)$. Next, we take the characterization of maximal cliques that can be leaves of some clique trees into consideration.

Lemma 5. *A maximal clique C can be a leaf of a clique tree if and only if C satisfies (1) C is incident to at most one edge in $\mathcal{E}_i$, and (2) C is not a cut vertex in $\mathcal{CG}(G)^*$.*

Proof. First, we suppose that C is a leaf of a clique tree $\mathcal{T}$. Since $\mathcal{T}$ is a clique tree of G, $\mathcal{T}$ is a spanning tree in $\mathcal{CG}(G)^*$ that includes all the edges in $\mathcal{E}_i$. Since C is a leaf of $\mathcal{T}$, C is incident to at most one edge of $\mathcal{E}_i$, and C is not a cut vertex of $\mathcal{CG}(G)^*$. Thus, C satisfies the conditions.

We next suppose that C satisfies the conditions. We assume that $\mathcal{CG}(G)$ contains two or more nodes. We choose any edge e from $\mathcal{E}_i \cup \mathcal{E}_d$ that is incident to C. We always can choose e since $\mathcal{CG}(G)$ is connected. Then, we remove C from $\mathcal{CG}(G)^*$. Since C is not a cut vertex, the resultant graph $\mathcal{CG}(G)'$ is still connected. Therefore, $\mathcal{CG}(G)'$ has a spanning tree $\mathcal{T}'$ which contains all the edges in $\mathcal{E}_i \setminus \{e\}$. Then, by adding e to $\mathcal{T}'$, we have a spanning tree $\mathcal{T}$ that contains all the edges in $\mathcal{E}_i$, and C is a leaf of $\mathcal{T}$. This concludes the proof. □

Hereafter, the pair of two conditions in Lemma 5 is said to be a *leaf condition*. A perfect sequence is obtained by removing a leaf of a clique tree $\mathcal{T}$. Thus, any perfect sequence is obtained by iteratively removing the maximal cliques satisfying a leaf condition. The converse is shown by the following lemma.

Lemma 6. *Any maximal clique sequence $\mathcal{S} = (C_1, \ldots, C_k)$ obtained by iteratively removing a maximal clique satisfying the leaf condition is a perfect sequence.*

Proof. Let $\mathcal{T}_1$ be a tree in $\mathcal{CG}(G)^*$ that consists of one vertex C_1, and $\mathcal{T}_i$ be a tree in $\mathcal{CG}(G)^*$ obtained by adding C_i to $\mathcal{T}_{i-1}$ and an edge e of $\mathcal{CG}(G)^*$ connecting C_i and a vertex in $\mathcal{T}_{i-1}$. If there is an edge of $\mathcal{E}_i$ connecting C_i and a vertex in $\mathcal{T}_{i-1}$, we choose the edge as e. Note that there is at most one such edge by the leaf condition. We observe that any C_i is a leaf of $\mathcal{T}_i$, and $\mathcal{T}_{i-1}$ is obtained by removing a leaf C_i from $\mathcal{T}_i$. Thus, $\mathcal{S} = (C_1, \ldots, C_k)$ is a perfect sequence. □

This lemma ensures that by repeatedly removing maximal cliques satisfying the leaf condition, we can obtain any perfect sequence. This yields the following algorithm to enumerate all perfect sequences.

Algorithm 3: All Perfect Sequences

Input : Chordal graph $G = (V, E)$;
Output: All perfect sequences of G;
1 construct $\mathcal{CG}(G)$;
2 find maximum weighted spanning tree $\mathcal{T}^*$ of $\mathcal{CG}(G)$;
3 by using $\mathcal{T}^*$, compute sets $\mathcal{E}_u$, $\mathcal{E}_i$, $\mathcal{E}_d$ of unnecessary, indispensable, and dispensable edges, respectively;
4 set P to empty sequence*[r]keep current perfect sequence let $\mathcal{CG}(G)^* := (\mathcal{C}(G), \mathcal{E}_i \cup \mathcal{E}_d)$;
5 call `Enumerate`$(\mathcal{CG}(G)^*, P)$;

Procedure `Enumerate`$(\mathcal{CG}(G)^* = (\mathcal{C}(G), \mathcal{E}_i \cup \mathcal{E}_d), P)$

Output: A perfect sequence at the last node;
7 $\mathcal{C}(G)$ contains one node C **output** $(C + P)$*[r]$C + P$ denotes concatenation of node C and sequence P compute $S := \{C \in \mathcal{C}(G) \mid C$ satisfies the leaf condition$\}$;
8 $C \in S$ call `Enumerate`$(\mathcal{CG}(G) \setminus C, C + P)$;

A part of the computation tree for the chordal graph in Fig. 1(a) is given in Fig. 2. The unweighted graph $\mathcal{CG}(G)^*$ contains two indispensable edges, $\{C_2, C_3\}$ and $\{C_4, C_5\}$. Now we are ready to show the main theorem in this paper.

Theorem 1. *For any chordal graph $G = (V, E)$, with $O(|V|^3)$ time and $O(|V|^2)$ space pre-computation, all perfect sequences can be enumerated in $O(1)$ time per sequence on average and $O(|V|^2)$ space.*

Proof. From Lemmas 5 and 6, we can see that Algorithm 3 generates all perfect sequences. Since each iteration adds a maximal clique on the top of the sequence when it generates a recursive call, no two iterations can output the same perfect sequence. Thereby any perfect sequence is generated exactly once. This shows the correctness of the algorithm.

Therefore, we concentrate on the analysis of the time complexity. The space complexity is easy to see. We first observe that the computation of set S in step 10 takes $O(n^2)$ time, where $n = |\mathcal{C}(G)|$ in the procedure. Therefore, the time complexity of each procedure call of `Enumerate` can be bounded above by cn^2 time for a positive constant c except for the computation time spent for the recursive calls generated by the procedure.

Now, a procedure call of `Enumerate` where $\mathcal{CG}(G) = (\mathcal{C}(G), \mathcal{E}_i \cup \mathcal{E}_d)$ is called a *k-level call* if $|\mathcal{C}(G)| = k$. Let $t(k)$ be the total computation time for k-level calls. When $k = 1$, we have a perfect sequence for each call. Therefore, we have $t(1) \leq cN$, where N is the number of perfect sequences.

When $k > 1$, there are at least two maximal cliques in $\mathcal{C}(G)$ satisfying the leaf condition. Therefore, there are at least two cliques in S, and the number of k-level calls is at most half of the number of $(k-1)$-level calls. Thus, we have $t(k) \leq \frac{k^2}{2^{k-1}} cN$ if $k > 1$. We can see that $t(2)+t(3)+t(4) \leq (2+\frac{9}{4}+2)cN = \frac{25}{4}cN$. For any $k > 4$, we have $\frac{k^2}{2^{k-1}} \leq \frac{8}{9}\frac{(k-1)^2}{2^{k-2}}$. Thus, we have $\sum_{k=5}^{\infty} t(k) \leq (\frac{8}{9} + (\frac{8}{9})^2 + \cdots)t(4) < 8t(4) \leq 16cN$. Therefore, each perfect sequence can be obtained on average in $O(1)$ time. □

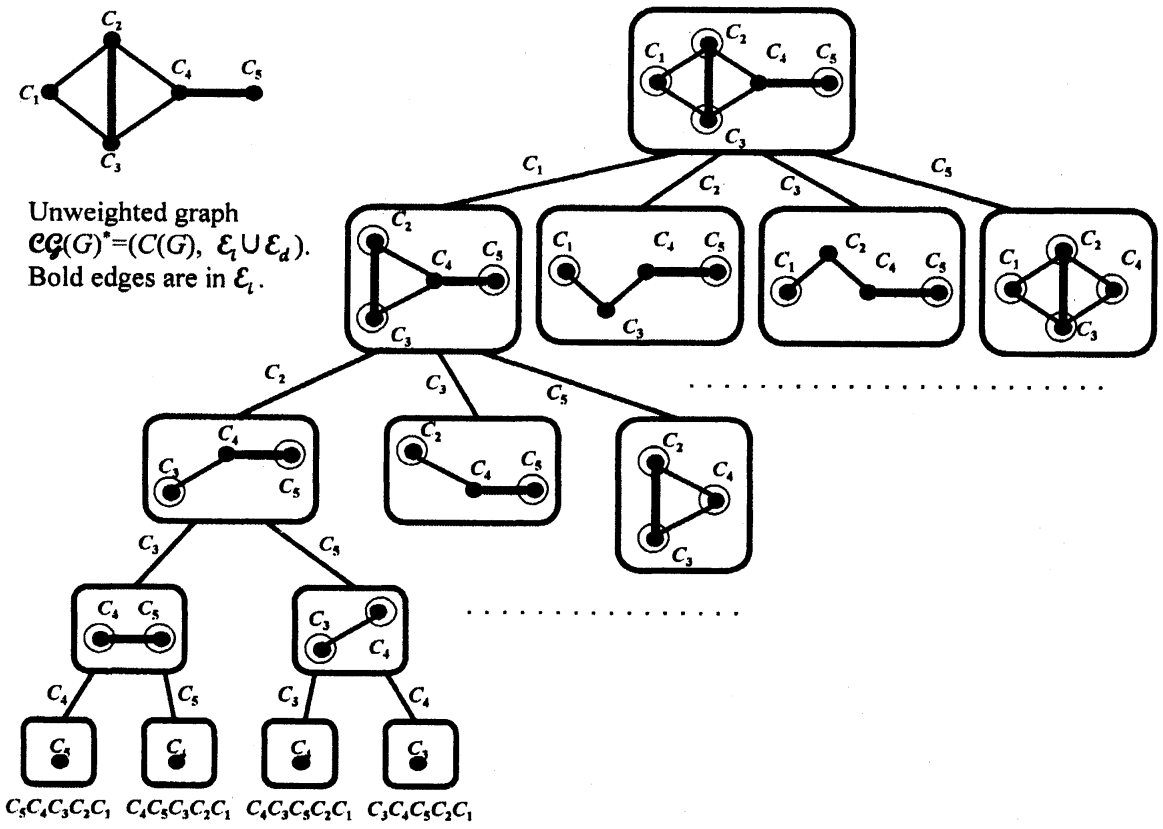

Fig. 2. Part of computation tree that enumerates all perfect sequences

We note here that we can include the time to output in $O(1)$ time for each perfect sequence on average. The first idea is to output the difference of the previous output. The second idea is to output at each step when the clique of the sequence is found. More precisely, we replace step 8 of `Enumerate` by the following three steps

output $+C$;
output "end of reverse of a perfect sequence";
output $-C$;

and replace step 12 of `Enumerate` by the following three steps

output $+C$;
call `Enumerate`$(\mathcal{CG}(G) \setminus C, C + P)$;
output $-C$;

Then we will incrementally have all perfect sequences, and the time complexity is still $O(1)$ time for each perfect sequence on average.

We also note that the *delay* in Algorithm 3, which is the maximum time between two consecutive perfect sequences, is $O(|V|^3)$ with a straightforward implementation. This time complexity comes from (1) the distance between two perfect sequences in the computation tree is $O(n)$ and (2) each computation

of the set S takes $O(n^2)$ time in step 10. However, we do not need to compute S completely every time. We can update S incrementally for each removal and addition with a suitable data structure, and the computation time can be reduced to $O(n)$ time. Hence it is not difficult to reduce the delay to $O(n^2)$, but the details are omitted here.

4 Conclusion

We propose an algorithm to enumerate all the perfect sequences of a given chordal graph. The time complexity for each perfect sequence is $O(1)$, which is the optimal time complexity. From the proof of the main theorem, we can see that the number of perfect sequences might be exponential in the size of the graph in general, and thus for large graphs the algorithm is impractical. Therefore, one of our future works is to construct an efficient random sampling algorithm. Our approach does not use clique trees, and thus, with polynomial time convergence, there is the possibility for efficient sampling.

References

1. J. R. S. Blair and B. Peyton. An Introduction to Chordal Graphs and Clique Trees. In *Graph Theory and Sparse Matrix Computation*, volume 56 of *IMA*, pages 1–29. (Ed. A. George and J.R. Gilbert and J.W.H. Liu), Springer, 1993.
2. A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.
3. L. S. Chandran, L. Ibarra, F. Ruskey, and J. Sawada. Generating and characterizing the perfect elimination orderings of a chordal graph. *Theoretical Computer Science*, 307:303–317, 2003.
4. J. Dahl, L. Vandenberghe, V. Roychowdhury. Covariance Selection for Non-chordal Graphs via Chordal Embedding. *Optimization Methods and Software*, 23(4):501–520, 2008.
5. M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Annals of Discrete Mathematics 57. Elsevier, 2nd edition, 2004.
6. H. Hara and A. Takemura. Boundary cliques, clique trees and perfect sequences of maximal cliques of a chordal graph. METR 2006-41, Department of Mathematical Informatics, University of Tokyo, 2006.
7. H. Hara and A. Takemura. Bayes Admissible Estimation of the Means in Poisson Decomposable Graphical Models. *Journal of Statistical Planning and Inference*, in press, 2008.
8. H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM Journal on Computing*, 28(5):1906–1922, 1999.
9. S. L. Lauritzen. *Graphical Models*. Clarendon Press, Oxford, 1996.
10. Y. Matsui, R. Uehara, and T. Uno. Enumeration of Perfect Sequences of Chordal Graph. ISAAC 2008, Springer, *Lecture Notes in Computer Science*, 5369:859–870, 2008.
11. T.A. McKee and F.R. McMorris. *Topics in Intersection Graph Theory*. SIAM, 1999.
12. D.J. Rose, R.E. Tarjan, and G.S. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.
13. J.P. Spinrad. *Efficient Graph Representations*. American Mathematical Society, 2003.
14. 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 Journal on Computing*, 13(3):566–579, 1984.