PAPER Special Section on Foundations of Computer Science

Generating Chordal Graphs Included in Given Graphs

Masashi KIYOMI[†a)], *Nonmember and* **Takeaki UNO**[†b)], *Member*

SUMMARY A chordal graph is a graph which contains no chordless cycle of at least four edges as an induced subgraph. The class of chordal graphs contains many famous graph classes such as trees, interval graphs, and split graphs, and is also a subclass of perfect graphs. In this paper, we address the problem of enumerating all labeled chordal graphs included in a given graph. We think of some variations of this problem. First we introduce an algorithm to enumerate all connected labeled chordal graphs in a complete graph of n vertices. Next, we extend the algorithm to an algorithm to enumerate all labeled chordal graphs in a n-vertices complete graph. Then, we show that we can use, with small changes, these algorithms to generate all (connected or not necessarily connected) labeled chordal graphs in arbitrary graph. All our algorithms are based on reverse search method, and time complexities to generate a chordal graph are O(1), and also O(1) delay. Additionally, we present an algorithm to generate every clique of a given chordal graph in constant time. Using these algorithms we obtain combinatorial Gray code like sequences for these graph structures in which the differences between two consecutive graphs are bounded by a constant size.

key words: *chordal graph, enumeration, constant time*

1. Introduction

A chordal graph is an undirected graph in which every cycle with at least four edges has a chord. Here, a chord of a cycle is an edge not in the cycle and connecting two vertices of the cycle. Chordal graphs have been considered to be important in the sense of both theoretical and application aspects. The class of chordal graphs contains many popular graph classes such as trees, interval graphs, and split graphs [6]. Many polynomial time algorithms to solve a lot of problems on chordal graphs are known. Moreover, the class of chordal graphs is a subclass of the class of perfect graphs [8]. Chordal graphs have many applications, for example, to matrix computation [5] or to relational database [4]. Chordal graphs are sometimes called triangulated graphs since every cycle in them is divided into some triangles. Chordal graphs are also used for numerical computation of models, and modeling some systems in computer science and social sciences [21].

Chordal graphs have many good properties. One important property of them is that a chordal graph has at least one *simplicial vertex*, where a vertex is simplicial iff its neighbors induce a clique. Given a graph G with a simplicial vertex v, we define an elimination of v from G as removing the vertex v from the vertex set of G and removing the edges adjacent to v from the edge set of G. An elimination of a simplicial vertex from a chordal graph results another chordal graph, and the size of the vertex set of the chordal graph is exactly smaller than that of the original chordal graph. Thus, we can iteratively eliminate simplicial vertices from a chordal graph until the graph has no vertex. The vertex ordering along which we eliminate the simplicial vertices are called *perfect elimination ordering*. Given a general graph, we can find a perfect elimination ordering of the graph in the linear time of the graph size, if there exists at least one such ordering [15]. We can characterize chordal graphs by perfect elimination orderings; a graph is chordal if and only if it has a perfect elimination ordering. These properties of chordal graphs give simple and fast algorithms for many problems. In particular, we can solve some combinatorial problems on chordal graphs, such as maximum independent set and minimum vertex coloring, in polynomial time with simple combinatorial algorithms.

Recently, because of the increase of computation power, many problems in practice are solved with enumeration. The graphical structures are used to model the systems and objects in many scientific area. Enumeration is used for simulating and finding good properties and new knowledges in these models. For example, enumeration algorithms for graph structures such as labeled paths, labeled trees, labeled graphs are used in frequent pattern mining problems [1], [2], [9], [18]. In optimization, many branch and bound type algorithms for combinatorial optimization explicitly or implicitly use enumeration of feasible solutions. Branching processes essentially generate all of the objects in the feasible domain, while the bounding processes try to reduce the number of objects to generate. It is important not to generate the same objects redundantly wherever possible to keep the algorithms efficient, and we can often do this with the techniques of enumeration. Enumeration of vertices of a polytope is used to find an optimum solution maximizing complicated functions. In column generation algorithms and set covering approaches, solutions of a subproblem are enumerated, and an optimal solution is found by combining them. Enumeration is also used in many graph algorithms. Moreover, enumeration itself has theoretical interests, and many studies have been done [7].

In this paper, we focus on enumeration of chordal subgraphs included in a given graph. More precisely, we enumerate edge subsets of a given graph which are edge sets of chordal subgraphs of a given graph. Here we assume that

Manuscript received March 21, 2005.
Manuscript revised June 28, 2005.

[†]The authors are with National Institute of Informatics, 2–1–2 Hitotsubashi, Chiyoda-ku, Tokyo, 101–8430 Japan.
a) E-mail: masashi@grad.nii.ac.jp
b) E-mail: uno@nii.jp

DOI: 10.1093/ietisy/e89–d.2.763

every vertex has an index (label) different from the other vertices, and we can describe every edge by a pair of vertices. We simply call such edge subsets *labeled chordal graphs*. We also consider the enumeration of connected labeled chordal subgraphs.

We propose efficient algorithms for the problems. The first is an algorithm for enumerating all connected chordal graphs included in a complete graph. The second is an algorithm for enumerating all connected chordal graphs included in an arbitrary graph. We will show that we can easily modify these algorithms so that they enumerate all (not necessary connected) chordal graphs. We also propose an algorithm for enumerating all cliques in a chordal graph, which is used as a subroutine by the algorithms for chordal graphs.

The time complexity of these three algorithms are all O(1) for each output, that is, the computation time is linear in the number of output chordal graphs, or cliques. We note that we output each chordal graph or clique by the symmetric difference from the previous one. Moreover, we show these algorithms directly lead combinatorial Gray code like sequences of several kinds of chordal subgraphs of a graph, and of cliques in a chordal graph, such that any consecutive two graphs differ at most three edges or vertices. The delay between two successive outputs in our algorithms are also O(1).

Our algorithms to enumerate labeled chordal graphs are based on reverse search. Reverse search was originally developed to enumerate all vertices of a given polytope represented by the intersection of half spaces [3]. Many enumeration algorithms use reverse search to enumerate many kind of objects such as spanning trees, trees, plane graphs, maximal cliques, etc. [11]–[13], [17].

In the following sections, we describe our problems and algorithms in detail. Section 2 describes a framework of reverse search and our enumeration scheme for labeled chordal graphs. In Sect. 3, we describe our enumeration algorithms. In Sect. 4, we analyze the time complexity of our algorithms. We explain a way to bound the delay in constant, and combinatorial Gray codes of labeled chordal graphs and cliques in Sect. 5. Finally we conclude the paper in Sect. 6.

2. Enumeration Scheme

In this section, we explain a basic concept of reverse search, then describe our enumeration scheme for labeled chordal graphs.

2.1 Reverse Search

Let $\mathcal{F}$ be the set of objects which we want to enumerate. In our problem, $\mathcal{F}$ is the set of labeled chordal graphs included in a given graph. We define a parent–child relation by determining a parent for each object except for some specified objects called *root objects*. The definition of the parents has to satisfy that no object is a proper ancestor of itself, i.e., by iteratively moving from an object x to the parent of x, to the parent of the parent of x, and so on, we never come to the start object x again. The graph representation of the relation induces a set of disjoint rooted trees spanning all objects, in which paths from all leaves aim to the roots. We illustrate an example of the graph representation in Fig. 1. Each object to be enumerated is drawn by a point, and an object and its parent is connected by a directed arrow.

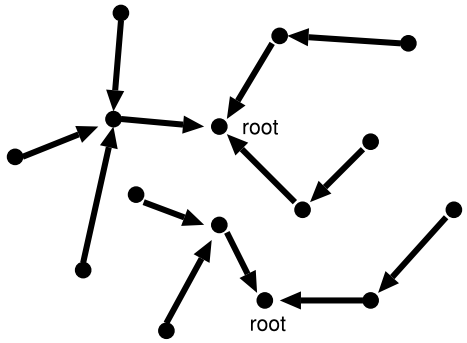


Fig. 1 Spanning forest on the objects to be enumerated, in which paths from all leaves aim to the roots.

Tracing each edge in the reverse direction enables us to perform depth first search to visit all objects. Thus, we can enumerate all objects using an algorithm to find all root objects, and an algorithm to find all children of an object. This is a basic concept of reverse search. Because of its simplicity and efficiency, reverse search has been used in a lot of algorithms [3], [11]–[13].

2.2 Parent–Child Relation

To construct a reverse search algorithm for labeled chordal graphs included in a graph, we need parent–child relations defined on the labeled chordal graphs.

Let $\bar{G} = (\bar{V}, \bar{E})$ be an arbitrary graph, and $\bar{V} = \{1, \ldots, n\}$. Suppose $G = (V, E)$ is a chordal subgraph of $\bar{G}$ and G has more than one edges. We define *minimum degree simplicial vertex* of G as the simplicial vertex having the minimum degree, and denote it by $s^*(G)$. If there are more than one such simplicial vertices, we choose the minimum (in vertices number) as $s^*(G)$, so that $s^*(G)$ is defined uniquely. Note that any chordal graph has at least one simplicial vertex, hence we can define $s^*(G)$ for any chordal graph. We define the parent of G as the graph obtained by elimination of $s^*(G)$ from G. Since an elimination of a simplicial vertex from a chordal graph results another chordal graph, the parent of G is also a chordal graph. We note that if G is connected, then the parent of G is also connected, since for any two neighbors of a simplicial vertex there is an edge which connect the vertices (See Fig. 2). The number of edges on the parent chordal graph is always strictly less than that of its child. Thus, no chordal graph becomes an ancestor of itself. Therefore, both the parent–child relation defined on all labeled chordal subgraphs of $\bar{G}$ and the parent–child relation defined on all connected labeled chordal subgraphs of $\bar{G}$ satisfy the conditions to be used in

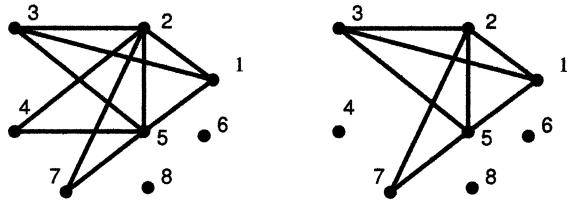


Fig. 2 The left chordal graph has simplicial vertices 1, 3, 4 and 7. The simplicial vertices of the minimum degree are 4 and 7. $s^*(G)$ is 4. The right chordal graph obtained by eliminating 4 from the left graph is the parent of the left graph.

reverse search described in the previous subsection. We illustrate an example of the parent–child relation of labeled chordal graphs in Fig. 2. Note that root chordal graphs are the graphs with exactly one edge.

3. Algorithms for Reverse Search

To enumerate all labeled chordal graphs included in a graph, we need an algorithm to enumerate all root labeled chordal graphs and also need an algorithm to enumerate all children of a given labeled chordal graph. The former algorithm is very simple; just generate all subgraphs having exactly one edge. In the following subsection, we describe the latter algorithm, the algorithm to generate children of labeled chordal graphs included in an arbitrary graph. We consider both the cases that the chordal graphs to be enumerated are connected, and not necessary to be connected.

3.1 Enumerating Children

The parent of a labeled chordal graph is obtained by eliminating a simplicial vertex. Hence, given a connected labeled chordal graph G in $\bar{G}$, any of its connected child is obtained by adding a vertex v to G. Here adding a vertex v to G means adding the vertex v to the vertex set of G and adding edges connecting the vertex v and some other vertices $C \subseteq V$ to the edge set E of G. It is necessary to obtain a child that C contains at least one vertex, and is a clique in G so that v is a simplicial vertex of the child. In the following, we characterize a necessary and sufficient condition for the resulting graph to be a child of G.

3.1.1 Characterization of Children of Connected Chordal Graph

In this subsection, we present a characterization of a child of a connected chordal graph in the parent–child relation of labeled connected chordal subgraphs of a given graph $\bar{G}$. We first introduce some notations. We denote the set of simplicial vertices in G by $S(G)$. The minimum degree in $S(G)$ is denoted by $k(G)$. We define $S_d(G)$ as the set of simplicial vertices of degree d in G, and particularly, we denote $S_{k(G)}(G)$ by $S^*(G)$. We denote the minimum vertex in a vertex set X by $\min(X)$. If $X = \emptyset$, we define $\min(X)$ is $+\infty$. Suppose $G = (V, E)$ be a connected labeled chordal graph

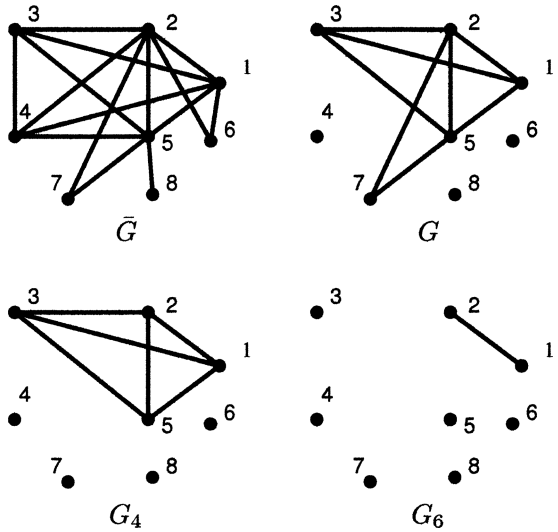


Fig. 3 Examples of $\bar{G}, G, G_4$ and G_6.

included in $\bar{G}$. Let v be a vertex of $\bar{G}$ and not in G. We denote by $N(G, v)$ the subgraph of G induced by the vertices which are adjacent to v in $\bar{G}$ (Fig. 3). We note that $N(G, v) = G$ holds for any G and v if $\bar{G}$ is a complete graph. Let C be a vertex subset in $N(G, v)$. We denote by $G(v, C)$ the graph obtained by adding v and edges connecting v and all vertices in C to G. It is necessary that any connected child H of a chordal graph G satisfies that $H = G(v, C)$ for some v and C, and C is a clique of G. We show below the necessary and sufficient condition for $H = G(v, C)$ to be a child of a chordal graph G.

Lemma 1: For a vertex $v \notin G$ and a clique C in $N(G, v)$, $G(v, C)$ is a child of G if and only if one of the following conditions holds.

(1) $|C| < k(G)$
(2) $|C| = k(G)$ and $v < \min(S^*(G) \setminus C)$
(3) $|C| = k(G) + 1$, $S^*(G) \subseteq C$ and $v < \min(S^*(G) \cup S_{k(G)+1}(G))$.

Proof: First, we claim the following propositions.

Proposition 1: Any simplicial vertex u ($\neq v$) in $G(v, C)$ is simplicial in G.

Proof: If u is not adjacent to v, the neighbors of u form a clique in G. If u is adjacent to v, the elimination of v from the neighbors of u also forms a clique in G. Thus, in both cases, u is simplicial in G. □

Proposition 2: $G(v, C)$ is a connected chordal graph, and v is simplicial in $G(v, C)$.

Proof: Let X be a cycle in $G(v, C)$ having at least four edges. If X does not include v, then X is included in G, hence X has a chord. If X includes v, then X includes at least two neighbors of v. The edge connecting the two neighbors is a chord of X, thus any cycle in $G(v, C)$ of at least four edges has a chord. Since the neighbors of v form C which is a clique, v is simplicial in $G(v, C)$. □

Now we prove the lemma. Above two propositions show that $G(v, C)$ is a connected child of G if and only if $s^*(G(v, C)) = v$. Thus, in order to prove the statement, we only need to check whether $s^*(G(v, C)) = v$ holds or not in the case of the conditions (1), (2) and (3). We consider the following four cases according to the size of C.

(a) $|C| < k(G)$: The degree of v in $G(v, C)$ is $|C|$ and is smaller than any other simplicial vertex in G. From Proposition 1, any simplicial vertex in $G(v, C)$ is a simplicial vertex in G, hence v is the unique minimum degree vertex among simplicial vertices in $G(v, C)$. Thus, $s^*(G(v, C)) = v$.

(b) $|C| = k(G)$: Similarly to the above, v has the minimum degree ($k(G)$) among simplicial vertices in $G(v, C)$. However it is possible that there are some other simplicial vertices whose degree are also $k(G)$. Since the degree of vertices of C in $G(v, C)$ is larger than $k(G)$, $S^*(G(v, C))$ is equal to $(S^*(G) \setminus C) \cup \{v\}$. Thus, $s^*(G(v, C)) = v$ if and only if v is smaller than $\min(S^*(G) \setminus C)$.

(c) $|C| = k(G) + 1$: In this case, v has the minimum degree in $S(G(v, C))$ if and only if $S^*(G) \subseteq C$ holds. Thus, $s^*(G(v, C)) = v$ if and only if $S^*(G) \subseteq C$ and $v < \min(S^*(G) \cup S_{k(G)+1}(G))$ hold.

(d) $|C| > k(G) + 1$: In this case, $C \cap S^*(G) = \emptyset$ since any vertex in $S^*(G)$ is adjacent to exactly $k(G)$ vertices. Thus, it is clear that $s^*(G(v, C))(= s^*(G))$ is never equal to v.

From these observations, we obtain that for a vertex v not in G and a clique C in G,

- if one of (1), (2) or (3) holds, $G(v, C)$ is a connected child of G,

and

- if none of (1), (2) and (3) holds, i.e., one of the below holds,
  - (2') $|C| = k(G)$ and $v > \min(S^*(G) \setminus C)$,
  - (3') $|C| = k(G) + 1$ and $S^*(G) \setminus C \neq \emptyset$,
  - (3") $|C| = k(G) + 1$ and $v > \min(S^*(G))$,
  - (4') $|C| > k(G) + 1$,

  then $G(v, C)$ is not a child of G.

Lemma 1 characterizes the children of a connected labeled chordal graph efficiently. From the lemma, we obtain an algorithm to enumerate the children of a connected labeled chordal graph. And it directly leads an algorithm to enumerate connected labeled chordal graphs in an arbitrary graph $\bar{G}$. In Fig. 4, we describe the algorithm. Note that it is easy to generate all cliques whose size are restricted, hence, if $\bar{G}$ is a complete graph, the algorithm becomes more simple.

3.1.2 Non-connected Chordal Subgraphs

To characterize the children for the parent–child relation on general labeled chordal graphs, we just modify the condition "C has to be a clique of $G = (V, E)$" to "C has to be a clique of G, or a singleton of a vertex not in $G \cup \{v\}$", which is equivalent to "C is a clique in the graph $(\bar{V}, E)$". We can prove the lemma below analogously. Note that we do not need to modify the definition of parents of chordal graphs.

Lemma 2: Let $G = (V, E)$ be a chordal subgraph of $\bar{G} = (\bar{V}, \bar{E})$. For a vertex $v \notin G$ and a clique C in $(\bar{V}, E)$, $G(v, C)$ is a child of G if and only if one of the following conditions holds.

(1) $|C| < k(G)$
(2) $|C| = k(G)$ and $v < \min(S^*(G) \setminus C)$
(3) $|C| = k(G) + 1$, $S^*(G) \subseteq C$ and $v < \min(S^*(G) \cup S_{k(G)+1}(G))$.

In the case of Lemma 1, it is clear that there is a bijection between $G(v, C)$ and a pair of (v, C). However, in the case of Lemma 2, if C is a singleton of a vertex $w \notin G \cup \{v\}$, we can sometimes swap v and w, that is, $G(w, \{v\})$ is also a child of G and $G(w, \{v\})$ is equal to $G(v, C)$. Thus, if we generate all children satisfying one of (1), (2) or (3), we sometimes generate the same child twice. To avoid this, we must not generate a child if C is a singleton w and w is smaller than v.

From the proofs of the lemmas, we directly obtain the following corollary.

Corollary 1: If $G(v, C)$ is a child of G, then $k(G(v, C)) = |C|$.

3.2 Enumerating Cliques in Chordal Graphs

The algorithm described in Fig. 4 requires a subroutine to

```
ALGORITHM Enum_Labeled_Connected_Chordal
(G = (V, E)):
1: output G;
2: if |V| = n then return;
3: if k(G) = 1 then do
     for each pair of vertices v ∉ G and u ∈ G
             such that (u, v) ∈ Ē and v < min(S*(G) \ {u}),
       call Enum_Labeled_Connected_Chordal(G(v, {u}));
     return;
   end;
4: for each d such that S_d(G) ≠ ∅,
     compute S_d(G) and min(S_d(G));
5: for each vertex v ∉ G with non-empty N(G, v),
     for each clique C in N(G, v) of size at most k(G) − 1
             in G,
       call Enum_Labeled_Connected_Chordal (G(v, C))
         for each v ∉ G;
6: for each pair of vertex v ∉ G and clique C in N(G, v)
             of size k(G)
             such that v < min(S*(G) \ C),
     call Enum_Labeled_Connected_Chordal (G(v, C));
7: for each pair of vertex v ∉ G and clique C in N(G, v)
             of size k(G) + 1
             such that S*(G) ⊆ C and
             v < min(S*(G) ∪ S_{k(G)+1}(G)),
     call Enum_Labeled_Connected_Chordal (G(v, C));
```

Fig. 4 Algorithm to enumerate connected labeled chordal graphs in an arbitrary graph.

enumerate cliques of sizes at most k in a given chordal graph. We can enumerate cliques in a general graphs in $O(|V|)$ time for each by simple backtracking algorithm. However, we can not use this algorithm for an enumeration algorithm which takes $O(1)$ time for each clique. Here, we describe a new algorithm to enumerate all size restricted cliques in a chordal graph $G = (V, E)$ which takes $O(1)$ time for each clique.

Let v be a simplicial vertex of G. We consider a partition of the set of cliques in G; all cliques including v, and all cliques not including v. Since the neighbors of v form a clique, v and its neighbors induce a complete graph. Thus, we can enumerate, as vertex set, every clique which includes v and whose size is up to k by combining v and every vertex subset whose size is at most $k-1$ and whose elements are neighbors of v. We can enumerate the cliques of size at most k and not including v recursively, that is, we can enumerate all cliques of size at most k in the graph obtained by eliminating v from G. We choose the vertex v in each level of the recursive calls along a perfect elimination ordering so that we need only constant time to obtain a simplicial vertex in each level of the recursive calls, if we already know a perfect elimination ordering of G. Note that we can generate each subset of size at most $k-1$ with reverse search, where we define the parent subset of a subset P including more than one elements as a subset obtained by removing the biggest element, as element number, from P. It is easy to obtain every child of a subset in constant time; We just add to a subset of size at most $k-2$ an element whose element number is bigger than those of all elements in it, then the resulting subset is a child of the subset, and all children are obtained in this way. Thus, we obtain the following theorems.

Theorem 1: We can enumerate all cliques of sizes at most k in a chordal graph in constant time for each clique on average and additional time to obtain a perfect elimination ordering, so that the size of the difference between consecutive two cliques is constant on average.

If we think the case that k is equal to n, we also obtain the following theorem.

Theorem 2: We can enumerate all cliques in a chordal graph in constant time for each clique on average, so that the size of the difference between consecutive two cliques is constant on average.

4. Time Complexity

In this section, we refer to the time complexity of our algorithms. First, we show our algorithm to enumerate all connected labeled chordal graphs in a complete graph costs constant time on average to enumerate every chordal graph. To show the time complexity to enumerate every labeled chordal graph in a complete graph is constant is analogously. Bounding the time complexity in the case of enumerating labeled chordal graphs in $\bar{G}$ which is not a complete graph needs some additional observations.

4.1 In Complete Graphs

Here we evaluate the computation time in the case that the input graph $\bar{G}$ is a complete graph.

We define an iteration of the algorithm as the operations in a vertex of the computation tree, which is a tree representation of the recursive structure of an execution of the algorithm. Thus, an iteration corresponds to the operations in an execution of ALGORITHM Enum_Connected_Labeled_Chordal excluding the operations in the recursive calls generated from it. Iterations and connected labeled chordal graphs have a one-to-one correspondence, thus we call an iteration inputting a chordal graph G *iteration of* G. In the rest of this subsection, we show the computation time of an iteration of G is linear in the number of children of G. To show this, we bound the computation time of each step in an iteration one by one.

It is clear that we can run step 1 and step 2 in Fig. 4 in constant time. From Corollary 1, we can compute $k(G)$ in constant time. Thus, we can perform the conditional branch in step 3 in constant time. In order to execute step 3 quickly in the case $k(G) = 1$, we maintain a sorted list of the vertices in $S_1(G)$ at every iteration. We can delete a vertex from the list in constant time. When the algorithm constructs $G(v, C)$ and adds v to $S_1(G)$ in some iteration, v is always smaller than any vertex in $S_1(G)$. Thus, we can add a vertex v to $S_1(G)$ in constant time by attaching v to the head of the list.

To compute $S_d(G)$ and $\min(S_d(G))$ in step 4, we take $O(|V|)$ time. When we execute step 4, we have $k(G) \geq 2$. Thus, the set of cliques of sizes at most $k(G)-1$ includes cliques whose sizes are one. We can also execute Step 7 in $O(|V|)$ time, since at most one clique satisfies the condition of step 7. Since the number of cliques of one vertex in G is $|V|$, G has at least $|V|$ children. Therefore, the computation time for steps 4 and 7 is linear in the number of children of G. This means that the computation time is linear in the number of children.

The enumeration algorithm of cliques in a chordal graph requires a perfect elimination ordering. Since in each iteration the algorithm adds a simplicial vertex to the graph, the ordering of the vertices added to obtain G reversely forms a perfect elimination ordering. Thus, we can keep a perfect elimination ordering of the current operating graph in memory, and update it in constant time at each iteration.

Step 6 takes long time in a straightforward way. To avoid this, we use cliques C' of size $k(G)-1$ found in step 5. We find all vertices $u \in G$ such that there is a vertex $v \notin G$ satisfying $v < \min(S^*(G) \setminus (C' \cup \{u\}))$. To satisfy the condition, $S^*(G) \setminus C'$ includes at most one vertex smaller than the minimum vertex not included in G. We can check this in $O(|C'|)$ time.

In the above, we saw that steps 5, 6 and the maintenance of the sorted list of the vertices in $S_1(G)$ take $O(|C|)$ time for each child. We can reduce the time to compute $G(v, C)$ in step 5 by using $G(v, C')$ where C' is the last clique obtained. By modifying $G(v, C')$ to obtain $G(v, C)$, we can

reduce the time complexity to $O((C \setminus C') \cup (C' \setminus C))$. Thus, from Theorem 1, the reduced computation time complexity is constant time for each on average. From similar observation, the computation time complexities for steps 5, 6 and the maintenance of the sorted list of the vertices in $S_1(G)$ are bounded by a constant for each child on average. Therefore, we obtain the following theorem.

Theorem 3: For a given complete graph K_n, we can enumerate all connected labeled chordal graphs, which are equivalent to all edge subsets of K_n inducing connected chordal graphs, in constant time for each edge subset, and in $O(n^2)$ memory.

In analogous way, we obtain the following theorem.

Theorem 4: For a given complete graph K_n, we can enumerate all labeled chordal graphs, which are equivalent to all edge subsets of K_n inducing chordal graphs, in constant time for each edge subset, and in $O(n^2)$ memory.

4.2 In Arbitrary Graphs

In the case of generating all connected or not necessarily connected labeled chordal graphs in an arbitrary graph $\bar{G}$, we have to compute $N(G, u)$ for all $u \in \bar{V} \setminus V$. Computing $N(G, u)$ for each u with no information takes long time. In order to reduce the time, we maintain $N(G, u)$ along the changes of the current chordal graph G. As a result, we will show that the computation time to obtain all $N(G', u)$ to be used is $O(|Chd(G')| + 1)$, where $Chd(G')$ is the set of children of G'. This implies that the computational time with respect to this operation is constant for each output chordal graph on average, since the sum of the number of children over all vertices in a tree is less than or equal to the number of vertices in the tree.

For each vertex not in G, let $M(G, v)$ be the set of vertices in $\bar{V} \setminus (V \cup \{v\})$ and adjacent to v. We keep $M(G, u)$ ($u \in \bar{V} \setminus V$) sorted in their indices. Suppose that in an iteration we obtain a child $G' = G(v, C)$ of G for some v and a clique C in $N(G, v)$. To generate a recursive call with respect to G', we compute $N(G', u)$ and $M(G', u)$ from $N(G, u)$ and $M(G, u)$ for each $u \in \bar{V} \setminus (V \cup \{v\})$. The computation with respect to $M(G', u)$ is $O(|M(G, v)|)$, since $M(G', u) = M(G, u) \setminus \{v\}$ if $u \in M(G, v)$ and $M(G', u) = M(G, u)$ otherwise.

For any vertex u in neither G nor $M(G, v)$, $N(G', u)$ is equal to $N(G, u)$. For any vertex $u \in M(G, v)$, $N(G', u)$ is obtained from $N(G, u)$ by adding v and edges connecting v and vertices both in $N(G, v)$ and C. In this way, the computation time to obtain $N(G', u)$ for all $u \in M(G, v)$ is $O(\sum_{u \in M(G,v)} |V(N(G, u))|)$, where $V(N(G, u))$ is the vertex set of $N(G, u)$. If $|C| \geq 2$, we obtain a child by adding an edge (u, w) to G' for any w in $N(G', u)$, since $k(G') \geq 2$ holds. Hence, we have

$$|M(G, v)| + \sum_{u \in M(G,v)} |V(N(G, u))| \leq O(|Chd(G')|).$$

Suppose that $|C| = 1$. We denote the unique element in C by w (i.e., $C = \{w\}$). In this case, $N(G', u)$ includes at most three edges not in $N(G, u)$, which are (u, v), (u, w) and (v, w). Thus, we can obtain $N(G', u)$ from $N(G, u)$ in constant time by looking the adjacency matrix of $\bar{G}$. The computation time to obtain $N(G', u)$ for all $u \in M(G, v)$ is $O(|M(G, v)|)$. We first consider the case that $|S_1(G')| = 1$, that is, v is the unique simplicial vertex in G' whose degree is one. In this case, for each $u \in M(G, v)$, G' has a child obtained by adding the edge (u, v) to G'. Hence, $|M(G, v)| = O(|Chd(G')|)$, and the time to compute all $N(G', u)$ for all $u \in M(G, v)$ is $O(|Chd(G')|)$.

Next we consider the case that $|S_1(G')| > 1$. Then, v is the minimum vertex among $S_1(G')$. Let v' be the second minimum in $S_1(G')$. We have the following property.

Lemma 3: For any descendant H of G', $s^*(H) < v'$, the degree of $s^*(H)$ is one, and $S_1(H)$ includes at least two vertices no greater than v' which are not adjacent to each other.

Proof: Suppose that H does not satisfy the condition, and without loss of generality, any ancestor of H satisfies the condition. Let $P(H)$ be the parent of H. From the assumption, $s^*(P(H)) < v'$, the degree of $s^*(P(H))$ is one, and $S_1(P(H))$ includes at least two vertices not greater than v' which are not adjacent to each other. Then, $s^*(H)$ has to be connected to at least two vertices in $S_1(P(H))$ which are not adjacent to each other. It contradicts the fact that the neighbors of $s^*(H)$ form a clique. □

From the lemma, we can see that for any descendant of G', $u > v'$ never be added. Thus, we do not need to compute $N(G, u)$ for any $u > v'$. We compute $N(G', u)$ and $M(G', u)$ for all vertices $u \in M(G, v)$ with $u < v'$. We can do this in constant time for each $u \in M(G, v)$ with $u < v'$ by tracing $M(G, v)$ in the ascending order. For each $u \in M(G, v)$ with $u' > v$, G' has a child obtained by adding the edge (u, v) to G'. Thus, $|\{u | u \in M(G, v), u < v'\}| = O(|Chd(G')|)$.

In every case, we have proved that the computation time to obtain $N(G', u)$ and $M(G', u)$ is $O(|Chd(G')| + 1)$. Thus, we obtain the following theorem.

Theorem 5: For a given graph $\bar{G}$, we can enumerate all labeled chordal graphs, which are equivalent to all edge subsets of $\bar{G}$ inducing chordal graphs, in constant time for each. The memory space necessary to run the algorithm is $O(n^3)$.

5. Constant Delay and Combinatorial Gray Codes

A combinatorial Gray code is an ordering of subset family in which consecutive two sets differ in small size [16]. Generally, the differences are minimal, but sometimes are not. In this section, we consider combinatorial Gray code like sequences of all chordal subgraphs and cliques so that any consecutive two differ at most three edges or three vertices. The differences are small, but possibly not minimal. Thus here we call them combinatorial Gray code like sequences.

In our algorithm, enumeration of cliques in a given chordal graph is composed of enumeration of all subsets of

a set. Thus, we can get a combinatorial Gray code by combining combinatorial Gray codes of these subsets. We can also see that the delay of the algorithm is constant. Similarly, we can obtain a combinatorial Gray codes for cliques of sizes at most a certain constant in a chordal graph.

Next we consider labeled chordal graphs. According to an algorithm described in Nakano and Uno [14], [20], if we have an enumeration algorithm traversing a search tree such that any parent and its child differ in constant size, the maximum difference between two consecutive outputs can be bounded in a constant size only with a modification on timing of output. The modification is that at the odd level of the recursion we output the objects before making recursive calls, and at the even level of the recursion, we output after the terminations of the recursive calls. In this way, at least one of three iterations outputs an object when the algorithm ascends or descends the search tree. Thus, the sequences of the output objects are combinatorial Gray codes like sequences. In this way, if each iteration takes constant time to make a recursive call, the delay is also constant time.

However, in our search tree on labeled chordal graphs, differently from that of subsets of size at most k, the size of difference between a parent and a child is not always in a constant size. In order to reduce the differences between consecutive chordal graphs, we enumerate all children of a labeled chordal graph G in an ordering that the differences between (a) G and the first child, (b) any two consecutive children, and (c) the last child and G are bounded by constant. This enumeration is equivalent to generating a combinatorial Gray code for cliques in a chordal graph. Using this enumeration method of children, the algorithm adds or deletes a constant number of edges to make a recursive call. Thus, we obtain combinatorial Gray code like sequences for them.

Theorem 6: There are algorithms for enumerating chordal subgraphs in a graph, connected chordal subgraphs in a graph, and cliques in a chordal graphs in constant delay, respectively.

Theorem 7: There is a sequence of cliques of a chordal graph such that two consecutive cliques differ at most three vertices.

Theorem 8: There are sequences of chordal subgraphs in a graph, and connected chordal subgraphs in a graph, respectively, such that two consecutive graphs differ at most three edges.

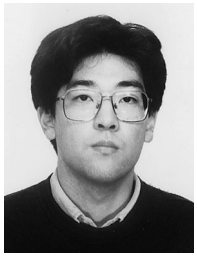
6. Conclusion

In this paper, we focused on the problems of enumerating all labeled chordal graphs on a vertex labeled graph. We introduced a good search tree on the set of (connected, or not necessarily connected) labeled chordal graphs, and presented efficient algorithms taking constant time on average for each chordal graph. As a corollary, we showed that we can enumerate all cliques in a chordal graph in constant time for each. Further, we showed the algorithms generate combinatorial Gray codes like sequences such that any two consecutive outputs differ at most three vertices or three edges. We leave the following problem as an open problem.

- Is there an output polynomial time algorithm to enumerate all maximal connected labeled chordal subgraph of a given general graph?

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," Proc. VLDB '94, pp.487–499, 1994.
[2] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa, "Efficient substructure discovery from large semi-structured data," Proc. Second SIAM International Conference on Data Mining 2002, pp.158–174, 2002.
[3] D. Avis and K. Fukuda, "Reverse search for enumeration," Discrete Appl. Math., vol.65, pp.21–46, 1996.
[4] C. Beeri, R. Fagin, D. Maier, and M. Yanakakis, "On the desirability of acyclic database schemes," J. ACM, vol.30, pp.479–513, 1983.
[5] J.R.S. Blair and B. Peyton, "An introduction to chordal graphs and clique trees," Graph Theory and Sparse Matrix Computation, vol.IMA56, pp.1–29, 1993.
[6] A. Brandstädt, V.B. Le, and J.P. Spinrad, "Graph classes: A survey," SIAM, 1999.
[7] L.A. Goldberg, Efficient algorithms for listing combinatorial structures, Cambridge University Press, New York, 1993.
[8] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Annals of Discrete Mathematics 57, 2nd ed., Elsevier, 2004.
[9] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," Lecture Notes in Computer Science 1910, 2000.
[10] D.S. Johnson, M. Yanakakis, and C.H. Papadimitriou, "On generating all maximal independent sets," Information Proceeding Letters, vol.27, pp.119–123, 1998.
[11] K. Makino and T. Uno, "New algorithms for enumerating all maximal cliques," Lecture Notes in Computer Science 3111, pp.260–272, 2004.
[12] S. Nakano, "Enumerating floorplans with n rooms," Lecture Notes in Computer Science 2223, pp.107–115, 2001.
[13] S. Nakano, "Efficient generation of triconnected plane triangulations," Computational Geometry Theory and Applications, vol.27, no.2, pp.109–122, 2004.
[14] S. Nakano and T. Uno, "Constant time generation of trees with specified diameter," Lecture Notes in Computer Science 3353, pp.33–45, 2004.
[15] D.J. Rose, R.E. Tarjan, and G.S. Lueker, "Algorithmic aspects of vertex elimination on graphs," SIAM J. Comput., vol.5, no.2, pp.266–283, 1976.
[16] C. Savage, "A survey of combinatorial gray codes," SIAM Review, vol.39, pp.605–629, 1997.
[17] A. Shioura, A. Tamura, and T. Uno, "An optimal algorithm for scanning all spanning trees of undirected graphs," SIAM J. Comput., vol.26, no.3, pp.678–692, 1997.
[18] K. Taniguchi, H. Sakamoto, H. Arimura, S. Shimozono, and S. Arikawa, "Mining semi-structured data by path expressions," Lecture Notes in Artificial Intelligence 2226, pp.378–388, 2001.
[19] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, "A new algorithm for generating all the maximal independent sets," SIAM J. Comput., vol.6, pp.505–517, 1977.
[20] T. Uno, "Two general methods to reduce delay and change of enumeration algorithms," National Institute of Informatics (in Japan) Technical Report, 004E, 2003.
[21] J. Whittaker, Graphical Models in Applied Multivariate Statistics, Wiley, Chichester, 1991.



Masashi Kiyomi was born in 1976. 2002, Master of Engineering, University of Tokyo, 2003-, doctor course student of National Institute of Informatics (Uno laboratory).

Takeaki Uno was born in 1970. 1988, Doctor of Science, Tokyo Institute of Technology, 1998–2001, assistant professor of Department Industrial Management and Engineering, Tokyo Institute of Technology. 2001–, associate professor of National Institute of Informatics. Research topic: algorithm theory and its applications for other science areas , especially, discrete algorithms and enumeration algorithms.