Title	Linear-time counting algorithms for independent sets in chordal graphs
Author(s)	Okamoto, Y; Uno, T; Uehara, R
Citation	Lecture Notes in Computer Science : including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, 3787: 433-444
Issue Date	2005
Type	Journal Article
Text version	author
URL	http://hdl.handle.net/10119/3276
Rights	This is the author-created version of Springer Berlin / Heidelberg, Yoshio Okamoto, Takeaki Uno and Ryuhei Uehara, Lecture Notes in Computer Science(Graph-Theoretic Concepts in Computer Science), 3787, 2005, 433-444. The original publication is available at www.springerlink.com, http://www.springerlink.com/content/768430570227u750
Description	

Linear-Time Counting Algorithms for Independent Sets in Chordal Graphs

Yoshio Okamoto[1], Takeaki Uno[2], and Ryuhei Uehara[3]

[1] Department of Information and Computer Sciences, Toyohashi University of Technology,
Hibarigaoka 1-1, Tempaku, Toyohashi, Aichi 441-8580, Japan. E-mail:
`okamotoy@ics.tut.ac.jp`
[2] National Institute of Informatics, Hitotsubashi 2-1-2, Chiyoda-ku, Tokyo 101-8430, Japan.
E-mail: `uno@nii.jp`
[3] School of Information Science, JAIST, Asahidai 1-1 , Nomi, Ishikawa 923-1292, Japan.
E-mail: `uehara@jaist.ac.jp`

Abstract. We study some counting and enumeration problems for chordal graphs, especially concerning independent sets. We first provide the following efficient algorithms for a chordal graph: (1) a linear-time algorithm for counting the number of independent sets; (2) a linear-time algorithm for counting the number of maximum independent sets; (3) a polynomial-time algorithm for counting the number of independent sets of a fixed size. With similar ideas, we show that enumeration (namely, listing) of the independent sets, the maximum independent sets, and the independent sets of a fixed size in a chordal graph can be done in constant amortized time per output. On the other hand, we prove that the following problems for a chordal graph are #P-complete: (1) counting the number of maximal independent sets; (2) counting the number of minimum maximal independent sets. With similar ideas, we also show that finding a minimum weighted maximal independent set in a chordal graph is NP-hard, and even hard to approximate.

Keywords: chordal graph, counting, enumeration, independent set, NP-completeness, #P-completeness, polynomial time algorithm.

1 Introduction

How can we cope with computationally hard graph problems? There are several possible answers, and one of them is to utilize the special graph structures arising from a particular context. This has been motivating the study of special graph classes in algorithmic graph theory [3, 13]. This paper deals with counting and enumeration problems from this perspective. Recently, counting and enumeration of some specified sets in a graph have been widely investigated, e.g., in the data mining area. In general, however, from the graph-theoretic point of view, those problems are hard even if input graphs are quite restricted. For example, counting the number of independent sets in a planar bipartite graph of maximum degree 4 is #P-complete [21]. Therefore, we wonder what kind of graph structures makes counting and enumeration problems tractable.

In this paper, we consider chordal graphs. A *chordal graph* is a graph in which every cycle of length at least four has a chord. From the practical point of view, chordal graphs

Table 1. Summary of the results. We denote the number of vertices and edges by n and m respectively. The running times for enumeration algorithms refer to amortized time per output.

Chordal graphs	Counting	[ref.]	Enumeration	[ref.]
independent sets	$O(n+m)$	[this paper]	$O(1)$	[this paper]
maximum independent sets	$O(n+m)$	[this paper]	$O(1)$	[this paper]
independent sets of size k	$O(k^2(n+m))$	[this paper]	$O(1)$	[this paper]
maximal independent sets	#P-complete	[this paper]	$O(n+m)$	[7, 16]
minimum maximal independent sets	#P-complete	[this paper]		

have numerous applications in, for example, sparse matrix computation (e.g., see Blair & Peyton [2]), relational databases [1], and computational biology [4]. Chordal graphs have been widely investigated, and they are sometimes called triangulated graphs, or rigid circuit graphs (see, e.g., Golumbic's book [13, Epilogue 2004]). A chordal graph has various characterizations; for example, a chordal graph is an intersection graph of subtrees of a tree, and a graph is chordal if and only if it admits a special vertex ordering, called perfect elimination ordering [3]. Also, the class of chordal graphs forms a wide subclass of perfect graphs [13].

It is known that many graph optimization problems can be solved in polynomial time for chordal graphs; to list a few of them, the maximum weighted clique problem, the maximum weighted independent set problem, the minimum coloring problem [12], the minimum maximal independent set problem [8]. There are also parallel algorithms to solve some of these problems efficiently [14]. However, relatively fewer problems have been studied for enumeration and counting in chordal graphs; the only algorithms we are aware of are the enumeration algorithms for all maximal cliques [11], all maximal independent sets [7, 16], all minimum separators and minimal separators [5], and all perfect elimination orderings [6].

In this paper, we investigate the problems concerning the number of independent sets in a chordal graph. Table 1 lists the results of the paper. We first give the following efficient algorithms for a chordal graph; (1) a linear-time algorithm to count the number of independent sets, (2) a linear-time algorithm to count the number of maximum independent sets, and (3) a polynomial-time algorithm to count the number of independent sets of a given size. The running time of the third algorithm is linear when the size is constant. Note that in general counting the number of independent sets and the number of maximum independent sets in a graph is #P-complete [17], and counting the number of independent sets of size k in a graph is #W[1]-complete [9] (namely, intractable in a parameterized sense). Let us also note that the time complexity here refers to the arithmetic operations, not to the bit operations.

The basic idea of these efficient algorithms is to invoke a clique tree associated with a chordal graph and perform a bottom-up computation via dynamic programming on the clique tree. A clique tree is based on the characterization of a chordal graph as an intersection graph of subtrees of a tree. Since a clique tree can be constructed in linear time and the structure of clique tree is simple, this approach leads to simple and efficient algorithms for the problems above. However, a careful analysis is necessary to obtain the linear-time complexity.

Along the same idea, we can also enumerate all independent sets, all maximum independent sets, and all independent sets of constant size in a chordal graph in $O(1)$ amortized time per output.

On the other hand, we show that the following counting problems are #P-complete: (1) counting the number of maximal independent sets in a chordal graph, and (2) counting the number of minimum maximal independent sets in a chordal graph. Using a modified reduction, we furthermore show that the problem to find a minimum weighted maximal independent set is NP-hard. We also show that the problem is even hard to approximate. More precisely speaking, there is no randomized polynomial-time approximation algorithm to find such a set within a factor of $c \ln |V|$, for some constant c, unless $\mathsf{NP} \subseteq \mathsf{ZTIME}(n^{O(\log\log n)})$. This is in contrast with a linear-time algorithm by Farber that finds a minimum weighted maximal independent set in a chordal graph when the weights are 0 or 1 [8].

Due to space limitation, some proofs are omitted.

2 Preliminaries

In this article, we assume that the reader has a moderate familiarity with graph theory. This section aims at fixing the notation and introducing a chordal graph and concepts around that. Let $G = (V, E)$ be a graph, which we always assume to be simple and finite, and also we assume that graphs are connected without loss of generality. The *neighborhood* of a vertex v in a graph $G = (V, E)$ is the set $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$. For a vertex subset U of V, we denote by $N_G(U)$ the set $\{v \in V \mid v \in N(u) \text{ for some } u \in U\}$. If no confusion can arise we will omit the subscript G. We denote the closed neighborhood $N(v) \cup \{v\}$ by $N[v]$. A vertex set I is an *independent set* of G if any pari of vertices in I is not an edge of G, and a vertex set C is a *clique* if every pair of vertices in C is an edge of G. An independent set is *maximum* if it has the largest size among all independent sets. An independent set is *maximal* if none of its proper supersets is an independent set. An independent set is *minimum maximal* if it is maximal and has the smallest size among all maximal independent sets. A maximum clique, a maximal clique and a minimum maximal clique are defined analogously. An edge which joins 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.

To a chordal graph $G = (V, E)$, we associate a tree T, called a *clique tree* of G, satisfying the following two properties. (A) The nodes of T are the maximal cliques of G. (B) For every vertex v of G, the subgraph T_v of T induced by the maximal cliques containing v is a tree. (In the literature, the condition (A) is sometimes weakened as each node is a vertex subset of G.) 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. Some details are explained in books [3, 19].

3 Linear-Time Algorithm to Count the Independent Sets

In this section, we describe an algorithm for counting the number of independent sets in a chordal graph. The basic idea of our algorithm is to divide the input graph into subgraphs induced by subtrees of the clique tree. Any two of these subtrees share a vertex of a clique if they are disjoint in the clique tree. This property is very powerful for counting the number of independent sets since any independent set can include at most one vertex of a clique. We compute the number of independent sets including each vertex of the clique, or no vertex of the clique by using the recursions.

First, we introduce some notations and state some lemmas. Given a chordal graph $G = (V, E)$, we construct a clique tree T of G. We now pick up any node in the clique tree T, regard the node as the root of T, and denote it by K_r. This is what we call a *rooted clique tree*. For a maximal clique K in a chordal graph G and a rooted clique tree T of G, a maximal clique K' in G is a *descendant* of K (with respect to T) if K' is a descendant of K in T. For convenience, we consider K itself a descendant of K as well, and when no confusion arises we omit saying "with respect to T." Let $\textsc{prt}(K)$ be the parent of K in T. For convenience, we define $\textsc{prt}(K_r)$ by $\emptyset$. We denote by $T(K)$ the subtree of T rooted at the node corresponding to the maximal clique K. Let $G(K)$ denote the subgraph of G induced by the vertices included in at least one node in $T(K)$. Observe that $G(K)$ is a chordal graph of which $T(K)$ is a clique tree.

For a graph G, let $\mathcal{IS}(G)$ be the family of independent sets in G. For a vertex v, let $\mathcal{IS}(G, v)$ be the family of independent sets in G including v, i.e., $\mathcal{IS}(G, v) := \{S \mid S \in \mathcal{IS}(G), v \in S\}$. For a vertex set U, let $\overline{\mathcal{IS}}(G, U)$ be the family of independent sets in G including no vertex of U, i.e., $\overline{\mathcal{IS}}(G, U) := \{S \mid S \in \mathcal{IS}(G), S \cap U = \emptyset\}$.

Lemma 1. *Let G be a chordal graph and T be a rooted clique tree of G. Choose a maximal clique K of G, and let $K_1, \ldots, K_\ell$ be the children of K in T. (If K is a leaf of the clique tree, we set $\ell := 0$.) Furthermore let $v \in K$ and $S \subseteq V(G(K))$. Then, $S \in \mathcal{IS}(G(K), v)$ if and only if S is represented by the union of $\{v\}$ and $S_1, \ldots, S_\ell$ such that $S_i \in \mathcal{IS}(G(K_i), v)$ if v belongs to K_i, and $S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)$ otherwise. Furthermore, such a representation is unique.*

By a close inspection of the proof, we can observe that for every $i, j \in \{1, \ldots, \ell\}$, $i \neq j$, it holds that $V(G(K_i)) \setminus K$ is disjoint from $V(G(K_j)) \setminus K$. This property gives a nice decomposition of the problem into several independent parts, and enables us to perform the dynamic programming on a clique tree.

By similar discussion, we obtain the following lemma.

Lemma 2. *Let G be a chordal graph and T be a rooted clique tree of G. Choose a maximal clique K of G, and let $K_1, \ldots, K_\ell$ be the children of K in T. (If K is a leaf of the clique tree, we set $\ell := 0$.)*
1. We have $S \in \overline{\mathcal{IS}}(G(K), K)$ if and only if S is the union of $S_1, \ldots, S_l$ such that $S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)$. Furthermore, such a representation is unique.
2. For each $i \in \{1, \ldots, \ell\}$, we have $S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)$ if and only if S_i belongs either to $\mathcal{IS}(G(K_i), v)$ for some $v \in K_i \setminus K$ or to $\overline{\mathcal{IS}}(G(K_i), K_i)$. Furthermore, S_i belongs to exactly one of them.

From these lemmas, we have the following recursive equations for $\mathcal{IS}$.

Equations 1 *Let G be a chordal graph and T be a rooted clique tree of G. For a maximal clique K of G which is not a leaf of the clique tree, let $K_1, \ldots, K_\ell$ be the children of*

```
Algorithm 1: #IndSets
Input  : A chordal graph $G = (V, E)$;
Output: The number of independent sets in $G$;
construct a rooted clique tree $T$ of $G$ with root $K_r$;
call #IndSetsIter($K_r$);
return $\left|\overline{\mathcal{IS}}(G, K_r)\right| + \sum_{v \in K_r} |\mathcal{IS}(G(K_r), v)|$.
```

```
Procedure #IndSetsIter($K$)
Input  : A maximal clique $K$ of the chordal graph $G$;
if $K$ is a leaf of $T$ then
    set $\left|\overline{\mathcal{IS}}(G(K), K)\right| := 0$ and $|\mathcal{IS}(K, v)| := 1$ for each $v \in K$;
else
    foreach child $K'$ of $K$ do call #IndSetsIter($K'$);
    foreach child $K'$ of $K$ do compute $\left|\overline{\mathcal{IS}}(G(K'), K \cap K')\right|$ by
        $\left|\overline{\mathcal{IS}}(G(K'), K')\right| + \sum_{u \in K' \setminus K} |\mathcal{IS}(G(K'), u)|$ ;
    compute $\left|\overline{\mathcal{IS}}(G(K), K)\right|$ by $\prod_{K' \in \text{CHD}(K)} \left|\overline{\mathcal{IS}}(G(K'), K \cap K')\right|$;
    foreach $v \in K$ do  compute $|\mathcal{IS}(G(K), v)|$ by
        $\prod_{K' \in \text{CHD}(K), v \in K'} |\mathcal{IS}(G(K'), v)| \times \prod_{K' \in \text{CHD}(K), v \notin K'} \left|\overline{\mathcal{IS}}(G(K'), K \cap K')\right|$ .
```

Fig. 1. Algorithm to count the number of independent sets in a chordal graph.

K in T. Furthermore, let $v \in K$. Then, the following identities hold. (We remind that $\dot\cup$ means "disjoint union.")

$$\mathcal{IS}(G(K)) = \overline{\mathcal{IS}}(G(K), K) \,\dot\cup \dot{\bigcup_{v \in K}} \mathcal{IS}(G(K), v);$$

$$\mathcal{IS}(G(K), v) = \{S \cup \{v\} \mid S = \bigcup_{i=1}^{\ell} S_i, S_i \in \begin{Bmatrix} \mathcal{IS}(G(K_i), v) & \textit{if } v \in K_i \\ \overline{\mathcal{IS}}(G(K_i), K \cap K_i) & \textit{otherwise} \end{Bmatrix}\};$$

$$\overline{\mathcal{IS}}(G(K), K) = \{S \mid S = \bigcup_{i=1}^{\ell} S_i, S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)\};$$

$$\overline{\mathcal{IS}}(G(K_i), K \cap K_i) = \overline{\mathcal{IS}}(G(K_i), K_i) \,\dot\cup \dot{\bigcup_{u \in K_i \setminus K}} \mathcal{IS}(G(K_i), u) \quad \textit{for each } i \in \{1, \ldots, \ell\}.$$

These equations lead us to the algorithm in Fig. 1 to count the number of independent sets in a chordal graph. For a maximal clique K of a chordal graph G, we denote the set of children of K in a rooted clique tree of G by $\text{CHD}(K)$.

Theorem 1. *The algorithm* #IndSets *outputs the number of independent sets in a chordal graph $G = (V, E)$ in $O(|V| + |E|)$ time.*

4 Linear-Time Algorithm to Count the Maximum Independent Sets

In this section, we modify Algorithm #IndSets to count the number of maximum independent sets in a chordal graph. For a set family $\mathcal{S}$, we denote by $\max(\mathcal{S})$ the cardinality of a largest set in $\mathcal{S}$, and $\operatorname{argmax}(\mathcal{S})$ denotes the family of largest sets in $\mathcal{S}$. For a graph G, let $\mathcal{MIS}(G)$ be the family of maximum independent sets in G. For a vertex v, let $\mathcal{MIS}(G, v)$ be the family of maximum independent sets in G including v, i.e., $\mathcal{MIS}(G, v) := \{S \in \mathcal{MIS}(G) \mid v \in S\}$. For a vertex set U, let $\overline{\mathcal{MIS}}(G, U)$ be the family of maximum independent sets in G including no vertex of U, i.e., $\overline{\mathcal{MIS}}(G, U) := \{S \in \mathcal{MIS}(G) \mid S \cap U = \emptyset\}$.

From lemmas stated in the previous section and Equations 1, we immediately have the following equations.

Equations 2 *With the same set-up as Equations 1, the following identities hold.*

$$
\begin{aligned}
\mathcal{MIS}(G(K)) &= \operatorname{argmax}(\overline{\mathcal{MIS}}(G(K), K) \,\dot{\cup} \dot{\bigcup_{v \in K}} \mathcal{MIS}(G(K), v));\\
\mathcal{MIS}(G(K), v) &= \operatorname{argmax}(\{S \mid S = \bigcup_{i=1}^{\ell} S_i, S_i \in \begin{cases} \mathcal{MIS}(G(K_i), v) & \text{if } v \in K_i \\ \overline{\mathcal{MIS}}(G(K_i), K \cap K_i) & \text{otherwise} \end{cases}\});\\
\overline{\mathcal{MIS}}(G(K), K) &= \operatorname{argmax}(\{S \mid S = \bigcup_{i=1}^{\ell} S_i, S_i \in \overline{\mathcal{MIS}}(G(K_i), K \cap K_i)\});\\
\overline{\mathcal{MIS}}(G(K_i), K \cap K_i) &= \operatorname{argmax}(\overline{\mathcal{MIS}}(G(K_i), K_i) \,\dot{\cup} \dot{\bigcup_{u \in K_i \setminus K}} \mathcal{MIS}(G(K_i), u)).
\end{aligned}
$$

Since the sets of each family on the left hand side have the same size in each equation, the cardinality of the set can be computed in the same order as Algorithm #IndSets. For example, $\mathcal{MIS}(G(K))$ can be computed as follows.

1. Set $N := 0$ and $M := \max(\overline{\mathcal{MIS}}(G(K), K) \cup \bigcup_{v \in K} \mathcal{MIS}(G(K), v))$;
2. if the size of a member of $\overline{\mathcal{MIS}}(G(K), K)$ is equal to M, then $N := N + \left|\overline{\mathcal{MIS}}(G(K), K)\right|$;
3. for each $v \in K$, if the size of a member of $\mathcal{MIS}(G(K), v))$ is equal to M, then $N := N + |\mathcal{MIS}(G(K), v))|$;
4. output N.

In this way we have the following theorem.

Theorem 2. *The number of maximum independent sets in a chordal graph $G = (V, E)$ can be computed in $O(|V| + |E|)$ time.*

5 Efficient Algorithm to Count the Independent Sets of Size *k*

In this section, we modify Algorithm #IndSets to count the number of independent sets of size k. For a graph G and a number k, let $\mathcal{IS}(G; k)$ be the family of independent sets in G of size k. For a vertex v, let $\mathcal{IS}(G, v; k)$ be the family of independent sets in G of size k including v, i.e., $\mathcal{IS}(G, v; k) := \{S \in \mathcal{IS}(G; k) \mid v \in S\}$. For a vertex set U, let $\overline{\mathcal{IS}}(G, U; k)$ be the family of independent sets in G of size k including no vertex of U, i.e., $\overline{\mathcal{IS}}(G, U; k) = \{S \in \mathcal{IS}(G; k) \mid S \cap U = \emptyset\}$.

From lemmas stated in Section 3 and Equations 1, we immediately obtain the following equations.

Equations 3

$$\mathcal{IS}(G(K); k) = \overline{\mathcal{IS}}(G(K), K; k) \,\dot{\cup} \dot{\bigcup_{v \in K}} \mathcal{IS}(G(K), v; k);$$

$$\mathcal{IS}(G(K), v; k) = \{S \mid S = \bigcup_{i=1}^{\ell} S_i, |S| = k, S_i \in \begin{Bmatrix} \mathcal{IS}(G(K_i), v) & \textit{if } v \in K_i \\ \overline{\mathcal{IS}}(G(K_i), K \cap K_i) & \textit{otherwise} \end{Bmatrix}\};$$

$$\overline{\mathcal{IS}}(G(K), K; k) = \{S \mid S = \bigcup_{i=1}^{\ell} S_i, |S| = k, S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)\};$$

$$\overline{\mathcal{IS}}(G(K_i), K \cap K_i; k) = \overline{\mathcal{IS}}(G(K_i), K_i; k) \,\dot{\cup} \dot{\bigcup_{u \in K_i \setminus K}} \mathcal{IS}(G(K_i), u; k).$$

In contrast to Equations 1, the second and third equations of Equations 3 do not give a straightforward way to compute $|\mathcal{IS}(G(K), v; k)|$ and $\left|\overline{\mathcal{IS}}(G(K), K; k)\right|$, respectively, since we have to count the number of combinations of $S_1, \ldots, S_\ell$ which generate an independent set of size k. To compute them, we use a more detailed algorithm.

Here we only explain a method to compute $|\mathcal{IS}(G(K), v; k)|$ since $\left|\overline{\mathcal{IS}}(G(K), K; k)\right|$ can be computed in a similar way. Fix an arbitrary vertex $v \in K$. Then, according to v, we give indices to the children of K such that $K_1, \ldots, K_p$ include v and $K_{p+1}, \ldots, K_\ell$ do not. For $k' \le k$ and $\ell' \le p$, let $\textsc{Num}(\ell'; k') := \{S \mid S = \bigcup_{i=1}^{\ell'} S_i, S_i \in \mathcal{IS}(K_i, v), |S| = k'\}$. For $k' \le k$ and $\ell' \ge p+1$, let $\overline{\textsc{Num}}(\ell'; k') := \{S \mid S = \bigcup_{i=\ell'}^{\ell} S_i, S_i \in \mathcal{IS}(K_i, K_i \setminus K), |S| = k'\}$. Then, it holds that $|\mathcal{IS}(G(K), v; k)| = \sum_{h=0}^{k}(|\textsc{Num}(p; h)| \times \left|\overline{\textsc{Num}}(p+1; k-h)\right|)$.

For each ℓ' and k', $|\textsc{Num}(\ell'; k')|$ can be computed in $O(k \times p)$ time based on the following recursive equation:

$$\left|\textsc{Num}(\ell'; k')\right| = \begin{cases} \sum_{h=0}^{k'} |\textsc{Num}(\ell' - 1; h)| \times |\mathcal{IS}(G(K_{\ell'}), v; k' - h)| & \text{if } \ell' > 1, \\ |\mathcal{IS}(G(K_1), v; k')| & \text{otherwise.} \end{cases}$$

Similarly, $\left|\overline{\textsc{Num}}(\ell'; k')\right|$ can be computed in $O(k')$ time. The computation of $|\textsc{Num}(\ell'; k')|$ and $\left|\overline{\textsc{Num}}(\ell'; k')\right|$ for all combinations of ℓ' and k' can be done in $O(k^2|\textsc{chd}(K)|)$ time, thus we can count the number of independent sets of size k in a chordal graph in $O(k^2|V|^2)$ time. In the following, we reduce the computation time by the same technique used in the previous sections.

Observe that $\left|\overline{\mathcal{IS}}(G(K), K; k')\right| = \sum_{h=0}^{k'} \left|\overline{\textsc{Num}}(p; h)\right| \times \left|\overline{\textsc{Num}}(p+1; k'-h)\right|$, which gives $\left|\overline{\textsc{Num}}(p+1; k')\right| \times \left|\overline{\textsc{Num}}(p; 0)\right| = \left|\overline{\mathcal{IS}}(G(K), K; k')\right| - \sum_{h=1}^{k'} \left|\overline{\textsc{Num}}(p; h)\right| \times \left|\overline{\textsc{Num}}(p+1; k'-h)\right|$. This implies that we can compute $\left|\overline{\textsc{Num}}(k'; p+1)\right|$ from

$\left|\overline{\mathcal{IS}}(G(K), K; h)\right|$ and $\left|\overline{\textsc{Num}}(p; h)\right|$ in the increasing order of k'. The computation time for this task is $O(k \times p)$.

In summary, we can compute $|\mathcal{IS}(G(K), v; k')|$ for all $v \in K$ and $k' \in \{0, \dots, k\}$ in $O(k^2 \sum_{v \in K} |\{K' \in \textsc{chd}(K) \mid v \in K'\}|)$ time. Therefore, the total computation time over all iterations can be bounded in the same way as the above section, and we obtain the following theorem.

Theorem 3. *1. The number of independent sets of size k in a chordal graph $G = (V, E)$ can be computed in $O(k^2(|V| + |E|))$ time.*

2. The numbers of independent sets of all sizes from 0 to $|V|$ in a chordal graph $G = (V, E)$ can be simultaneously computed in $O(|V|^2(|V| + |E|))$ time.

6 Enumeration

Equations 1 in Section 3 directly give the following algorithm for enumerating the independent sets of a given chordal graph, in which each procedure corresponds to an equation of Equations 1.

```
Algorithm 3: EnumIS($G$)
Input  : a chordal graph $G = (V, E)$;
Output: all independent sets in $G$;
construct a clique tree $T$ of $G$ with root $K$;
foreach $u \in K$ do enumerate all independent sets in $\mathcal{IS}(G, u)$ by EnumIS2($K, u$);
enumerate all independent sets in $\overline{\mathcal{IS}}(G, K)$ by EnumIS3($K$).
```

```
Procedure EnumIS2($K, u$)
Input  : A maximal clique $K$ of $G$, a vertex $u \in K$;
if $K$ has no child then
    output $\{u\}$;  //output an independent set if the bottom level is reached
else
    foreach child $K_i$ of $K$ such that $u \in K_i$ do  enumerate all independent sets in
        $\mathcal{IS}(G(K_i), u)$ by EnumIS2($K_i, u$);
    foreach child $K_i$ of $K$ such that $u \notin K_i$ do enumerate all independent sets in
        $\overline{\mathcal{IS}}(G(K_i), K \cap K_i)$ by EnumIS4($K_i$);
    output all independent sets in $\mathcal{IS}(G(K), u)$ by combining the independent sets in
        $\mathcal{IS}(G(K_i), u)$ and in $\overline{\mathcal{IS}}(G(K_j), K \cap K_j)$ for all $i$, $j$;
```

```
Procedure EnumIS3($K$)
Input  : A maximal clique $K$ of $G$;
if $K$ has no child then
    output $\emptyset$;  //output an independent set if the bottom level is reached
else
    foreach child $K_i$ of $K$ do enumerate all independent sets in $\overline{\mathcal{IS}}(G(K_i), K \cap K_i)$ by
        EnumIS4($K_i$);
    output all independent sets in $\overline{\mathcal{IS}}(G(K), K)$ by combining the independent sets in
        $\overline{\mathcal{IS}}(G(K_i), K \cap K_i)$;
```

```
Procedure EnumIS4($K$)
    Input : A maximal clique $K$ of $G$;
15  call EnumIS3(K);
16  foreach $u \in K \setminus \textsc{prt}(K)$ do enumerate all independent sets in $\mathcal{IS}(G(K), u)$ by
        EnumIS2($G(K), u$);
17  output all independent sets in $\overline{\mathcal{IS}}(G(K), K \cap \textsc{prt}(K))$ by combining the independent sets
        in $\mathcal{IS}(G(K), u)$;
```

From the lemmas and theorems in the previous sections, EnumIS(G) surely enumerates all independent sets in G. However, we cannot bound its time complexity by constant for each output. In the following, we present a slight modification to obtain a constant-time enumeration algorithm.

Let us consider the computation tree of this algorithm. A *computation tree* is a rooted-tree representation of a recursive structure, in which the vertices are recursive calls, and the edges connect two vertices if and only if one vertex recursively calls the other. We define an *iteration* of the algorithm by the operations done in a vertex of the computation tree. In other words, an iteration is the computation in some procedure P recursively called by another procedure, in which the computation in the recursive calls generated by P is excluded.

We first reduce the number of iterations by the following two modifications. (1) If an iteration I generated by an iteration I_p recursively calls just one iteration I_c, we modify the algorithm so that I_p recursively calls I_c directly. (2) If an iteration I outputs just one independent set, merge I and the iteration which recursively calls I into one.

For a given chordal graph $G = (V, E)$ and a rooted clique tree of G, the number of possible inputs for each procedure is at most $O(|E|)$, as in our counting algorithms. Thus, we can enumerate all of these cases in $O(|E|)$ time, and keep the results of modifications (1) and (2) in the memory. It can be done as a preprocessing within $O(|E|)$ time.

By these modifications, we can see that any iteration which is a leaf of the computation tree outputs at least two independent sets, thus the number of iterations is not greater than the number of independent sets in G. We can also see that if an iteration outputs just one independent set, then, the input clique must be a leaf of the clique tree. Hence, the size of the output independent set is at most one.

We next consider how to compute all combinations of independent sets in, for example, Step 9 of the algorithm. In the procedures, the independent sets for K are generated by combining the independent recursive calls for several maximal cliques, say K_1 and K_2. This step can be implemented as follows. First, we compute an indenendent set I_1 for K_1, and for this I_1, we compute all independent sets I_2 for K_2, and output $I_1 \cup I_2$. Next we compute another independent set I'_1 for K_1, and compute all independent sets I_2 for K_2, and output $I_1 \cup I_2$, then compute yet another independent set for K_1, and so on. Then the computation time in one iteration is proportional to (the number of recursive calls generated) times (the maximum number of vertices added to the current independent set). Because of modification (2), any iteration adds at most one vertex to the current independent set. Therefore, the total time complexity of the algorithm is linear in the number of independent sets.

Theorem 4. *All independent sets in a chordal graph can be enumerated in constant time for each on average with additional* $O(|V| + |E|)$ *time for preprocessing.*

Similar algorithms can be developed to enumerate the maximum independent sets and the independent sets of size k. However, some iterations may add to the current independent set several vertices not bounded by a constant. Since there are at most $|E|$ kinds of inputs for each procedure, we can enumerate all such sets of vertices that will be added in an iteration, and put an identical name to each set of vertices in short time. By adding the name instead of adding vertices in a vertex set, we can execute the addition in constant time. Thus, the maximum independent sets and the independent sets of size k can be enumerated in constant time for each on average with additional $O((|V| + |E|)|V|^2)$ time for preprocessing.

7 Hardness of Counting the Maximal Independent Sets

In this section, we show the hardness results for counting the number of maximal independent sets in a chordal graph. Although finding a maximal independent set is easy even in a general graph, we show that the counting version of the problem is actually hard.

Theorem 5. *Counting the number of maximal independent sets in a chordal graph is* #P*-complete.*

The proof is based on a reduction from the counting problem of the number of set covers. Let X be a finite set, and $\mathcal{S} \subseteq 2^X$ be a family of subsets of X. A *set cover* of X is a subfamily $\mathcal{F} \subseteq \mathcal{S}$ such that $\bigcup \mathcal{F} = X$. Counting the number of set covers is #P-complete [17].

Proof of Theorem 5 (Sketch). The membership in #P is immediate. To show the #P-hardness, we use a polynomial-time reduction of the problem for counting the number of set covers to our problem.

Let X be a finite set and $\mathcal{S} \subseteq 2^X$ be a family of subsets of X, and consider them as an instance of the set cover problem. Let us put $\mathcal{S} := \{S_1, \ldots, S_t\}$. From X and $\mathcal{S}$, we construct a chordal graph $G = (V, E)$ in the following way.

We set $V := X \cup \mathcal{S} \cup \mathcal{S}'$, where $\mathcal{S}' := \{S'_1, \ldots, S'_t\}$. Namely, $\mathcal{S}'$ is a copy of $\mathcal{S}$. Now, we draw edges. There are three kinds of edges. (1) We connect every pair of vertices in X by an edge. (2) For every $S \in \mathcal{S}$, we connect $x \in X$ and S by an edge if and only if $x \in S$. (3) For every $S \in \mathcal{S}$, we connect S and S' (a copy of S) by an edge. Formally speaking, we define $E := \{\{x, y\} \mid x, y \in X\} \cup \{\{x, S\} \mid x \in X, S \in \mathcal{S}, x \in S\} \cup \{\{S, S'\} \mid S \in \mathcal{S}\}$. This completes our construction, which can be done in polynomial time. The constructed graph G is indeed chordal.

Now, we look at the relation between the set covers of X and the maximal independent sets of G. Let U be a maximal independent set of G. We distinguish two cases.

Case 1. Consider the case in which U contains a vertex $x \in X$. Let $G_x := G \setminus N_G[x]$. By the construction, we have that $V(G_x) = \{S \in \mathcal{S} \mid x \notin S\} \cup \mathcal{S}'$ and $E(G_x) = \{\{S, S'\} \mid S \in \mathcal{S}, x \notin S\}$. Then the number of maximal independent sets containing x is exactly $2^{|\{S \in \mathcal{S} | x \notin S\}|}$.

Case 2. Consider the case in which U contains no vertex of X. Then, the number of maximal independent sets containing no vertex of X is equal to the number of set covers of X.

To summarize, we obtained that the number of maximal independent sets of G is equal to the number of set covers of X plus $\sum_{x \in X} 2^{|\{S \in \mathcal{S} | x \notin S\}|}$. Since the last sum can be computed in polynomial time, this concludes the reduction. □

As a variation, let us consider the problem for counting the minimum maximal independent sets in a chordal graph. Note that a minimum maximal independent set in a chordal graph can be found in polynomial time [8]. In contrast to that, the counting version is hard.

Theorem 6. *Counting the minimum maximal independent sets in a chordal graph is* #P-*complete.*

8 Hardness of Finding a Minimum Weighted Maximal Independent Set

In this section, we consider an optimization problem to find a minimum weighted maximal independent set in a chordal graph. Namely, given a chordal graph G and a weight for each vertex, we are asked to find a maximal independent set of G with minimum weight. Here, the weight of a vertex subset is the sum of the weights of its vertices.

Notice that there is a linear-time algorithm for this problem when the weight of each vertex is zero or one [8]. On the contrary, we show that the problem is actually hard when the weight is arbitrary.

Theorem 7. *Finding a minimum weighted maximal independent set in a chordal graph is* NP-*hard.*

The proof is similar to what we saw in the previous section. We use the optimization version of the set cover problem, namely the minimum set cover problem. It is known that the minimum set cover problem is NP-hard.

Proof of Theorem 7. For a given instance of the minimum set cover problem, we use the same construction of a graph G as in the proof of Theorem 5. We define a weight function w as follows: $w(x) := 2|\mathcal{S}| + 1$ for every $x \in X$; $w(S) := 2$ for every $S \in \mathcal{S}$; $w(S') := 1$ for every $S' \in \mathcal{S}'$. This completes the construction.

Now, observe that $\mathcal{S}$ is a maximal independent set of the constructed graph G, and the weight of $\mathcal{S}$ is $2|\mathcal{S}|$. Therefore, no element of X takes part in any minimum weighted maximal independent set of G. Then, from the discussion in the proof of Theorem 5, if M is a maximal independent set of G satisfying $M \cap X = \emptyset$, then $M \cap \mathcal{S}$ is a set cover of X. The weight of M is $|M \cap \mathcal{S}| + |\mathcal{S}|$. Therefore, if M is a minimum weighted independent set of G, then M minimizes $|M \cap \mathcal{S}|$, which is the size of a set cover. Hence, $M \cap \mathcal{S}$ is a minimum set cover. This concludes the reduction. □

We can further show the hardness to get an approximation algorithm running in polynomial time. The precise statement is as follows (ZTIME(t) is the class of languages which have a randomized algorithm running in expected time t with zero error).

Theorem 8. *There is no randomized polynomial-time algorithm for the minimum weight maximal independent set problem in a chordal graph with approximation ratio $c \ln |V|$, for some fixed constant c, unless* NP $\subseteq$ ZTIME($n^{O(\log \log n)}$).

Acknowledgement The authors are grateful to L. Shankar Ram for pointing out a paper [5].

References

1. C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the Desirability of Acyclic Database Schemes. *J. of the ACM*, 30:479–513, 1983.
2. 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.
3. A. Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.
4. P. Buneman. A Characterization of Rigid Circuit Graphs. *Disc. Math.*, 9:205–212, 1974.
5. L.S. Chandran. A Linear Time Algorithm for Enumerating All the Minimum and Minimal Separators of a Chordal Graph. *COCOON*, pages 308–317. LNCS Vol. 2108, Springer-Verlag, 2001.
6. 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.
7. D. Eppstein. All Maximal Independent Sets and Dynamic Dominance for Sparse Graphs. *SODA*, pages 451–459, ACM, 2005.
8. M. Farber. Independent Domination in Chordal Graphs. *Operations Research Letters*, 1(4):134–138, 1982.
9. J. Flum and M. Grohe. The Parameterized Complexity of Counting Problems. *SIAM J. Comput.*, 33(4):892–922, 2004.
10. P. Frankl and R.M. Wilson. Intersection theorems with geometric consequences. *Combinatorica*, 1:357–368, 1981.
11. D.R. Fulkerson and O.A. Gross. Incidence Matrices and Interval Graphs. *Pacific J. Math.*, 15:835–855, 1965.
12. F. Gavril. Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph. *SIAM J. Comput.*, 1(2):180–187, 1972.
13. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Annals of Discrete Mathematics 57. Elsevier, 2nd edition, 2004.
14. P.N. Klein. Efficient Parallel Algorithms for Chordal Graphs. *SIAM J. Comput.*, 25(4):797–827, 1996.
15. V.S. Anil Kumar, Sunil Arya, and H. Ramesh. Hardness of Set Cover with Intersection 1. *ICALP*, pages 624–635. LNCS Vol. 1853, Springer-Verlag, 2000.
16. J.Y.-T. Leung. Fast Algorithms for Generating All Maximal Independent Sets of Interval, Circular-Arc and Chordal Graphs. *J. of Algorithms*, 5:22–35, 1984.
17. J.S. Provan and M.O. Ball. The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected. *SIAM J. Comput.*, 12:777–788, 1983.
18. D.J. Rose, R.E. Tarjan, and G.S. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM J. Comput.*, 5(2):266–283, 1976.
19. J.P. Spinrad. *Efficient Graph Representations*. American Mathematical Society, 2003.
20. R.E. Tarjan and M. Yannakakis. Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.
21. S.P. Vadhan. The Complexity of Counting in Sparse, Regular, and Planar Graphs. *SIAM J. Comput.*, 31(2):398–427, 2001.