Counting the Number of Independent Sets in Chordal Graphs[1]

Yoshio Okamoto[2], Takeaki Uno[3], Ryuhei Uehara[4]

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 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, 14]. 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 [22]. 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 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 [14, 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 [14].

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 [13], the minimum maximal independent set problem [9]. There are also parallel algorithms to solve

[1] An extended abstract of this paper appeared in the proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2005), pages 433-444, Lecture Notes in Compter Science Vol. 3787, Springer Verlag, 2005.
[2] Department of Information and Computer Sciences, Toyohashi University of Technology, Hibarigaoka 1-1, Tempaku, Toyohashi, Aichi 441-8580, Japan. okamotoy@ics.tut.ac.jp
[3] National Institute of Informatics, Hitotsubashi 2-1-2, Chiyoda-ku, Tokyo 101-8430, Japan. uno@nii.jp
[4] Corresponding author. School of Information Science, Japan Advanced Institute of Science and Technology, Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan. uehara@jaist.ac.jp

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 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]	$O(n+m)$ [8, 16]
minimum maximal independent sets	#P-complete [this paper]	

some of these problems efficiently [15]. 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 [12], all maximal independent sets [16] (see also conclusions in a paper of Eppstein [8]), 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 [18], and counting the number of independent sets of size k in a graph is #W[1]-complete [11] (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 a 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 $O(1)$ 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, there is no polynomial-time approximation algorithm to find such a set within a factor of $c \ln |V|$, for some constant c, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$. 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 [9].

The organization of the paper is as follows. Section 2 introduces the concept of a clique tree. In Section 3, we devise a linear-time algorithm for counting the number of independent sets, and in Section 4, we discuss how to count the maximum independent sets in linear time. In Section 5, we provide an efficient algorithm for counting the number of independent sets of each size simultaneously. In Section 6, we briefly describe how to apply our method for counting to enumeration, which leads to constant time algorithms. In Section 7, we prove that counting the number of maximal independent sets and counting the number of minimum maximal independent sets are hard. In Section 8, we modify the reduction in Section 7 to show that it is hard to find a minimum weighted maximal independent set, and even hard to approximate.

2 Preliminaries

A graph $G = (V, E)$ consists of a finite set V of *vertices* and a collection E of 2-element subsets of V called *edges*. The vertex set and the edge set of G are often denoted by $V(G)$ and $E(G)$ respectively. 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 $\deg_G(v)$. If no confusion can arise we will omit the subscript G. We denote the closed neighborhood $N_G(v) \cup \{v\}$ by $N[v]$. 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* if every pair of vertices in C is joined by an edge in G. We regard an empty set as an independent set of size zero. 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 four has a chord. Given a graph $G = (V, E)$, a vertex $v \in V$ is *simplicial* in G if $N_G(v)$ is a clique in G. An ordering $v_1, \dots, v_n$ of the vertices of V is a *perfect elimination ordering* of G if the vertex v_i is simplicial in $G[\{v_i, v_{i+1}, \dots, v_n\}]$ for all $i = 1, \dots, n$. It is known that a graph is chordal if and only if it has a perfect elimination ordering [3, Section 1.2]. Given a chordal graph a perfect elimination ordering of the graph can be found in linear time [19, 21].

To a chordal graph $G = (V, E)$, we associate a tree T, called a *clique tree* of G, satisfying the following three properties. (A) The nodes of T are the maximal cliques of G. (B) Two nodes of T are adjacent only if their intersection is non-empty. (C) 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 (not necessarily maximal) clique 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, 20]. The following property is important in the running time analysis of our algorithms.

Lemma 2.1. *Let* $G = (V, E)$ *be a chordal graph, and denote by* $\mathcal{K}$ *the family of maximal cliques of* G*. Then, it holds that* $\sum_{K \in \mathcal{K}} |K| = O(|V| + |E|)$.

Proof. Take any perfect elimination ordering $v_1, v_2, \dots, v_n$ of G. Let $C(v_i) := N_G[v_i] \cap \{v_i, v_{i+1}, \dots, v_n\}$. It is known that for every maximal clique K of G there exists a vertex $v_i \in V$ such that $K = C(v_i)$ holds [12]. Since $C(v_i) \subseteq N_G[v_i]$, we have $|C(v_i)| \le |N_G[v_i]| = 1 + \deg_G(v_i)$. Putting together, we obtain $\sum_{K \in \mathcal{K}} |K| \le \sum_{v \in V} |C(v)| \le \sum_{v \in V}(1 + \deg_G(v)) = |V| + 2|E| = O(|V| + |E|)$. □

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 G. 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. We also denote the set of children of K in T by $\textsc{chd}(K)$. For convenience, we define $\textsc{prt}(K_r) := \emptyset$ and $\textsc{chd}(K_\ell) := \emptyset$ for each leaf K_ℓ. 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.

The basic idea of our algorithm is to divide the input graph into subgraphs induced by subtrees of the (rooted) clique tree T. Let K be any maximal clique with two children K_1, K_2 on a rooted clique tree T. Let T_1 and T_2 be two node-disjoint subtrees of T which are rooted at K_1 and K_2, respectively. Let C be the set of vertices in G shared by T_1 and T_2. Then, C induces a clique, and $C \subset K$. This property is very useful for counting the number of independent sets since every independent set can contain at most one vertex of the clique C. Therefore we can partition the family of independent sets into two groups; a family of independent sets that contain one vertex from C, and the other family of independent sets that contain no vertex from C. Moreover, since $C \subset K$, $(K_1 \setminus K)$ and $(K_2 \setminus K)$ share no vertex. Thus, in each case, we can divide the counting problem onto two disjoint subgraphs $G(K_1)$ and $G(K_2)$. Hence we can use a recursive approach.

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 3.1. *Let G be a chordal graph and T be a rooted clique tree of G. Choose a maximal clique K of G, and let $K_1, \dots, K_\ell$ be the children in $\text{CHD}(K)$. 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 $S = \{v\} \cup S_1 \cup \cdots \cup S_\ell$ 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.*

Proof. We first show the only-if part. Assume that $S \in \mathcal{IS}(G(K), v)$. Let $S_i := S \cap G(K_i)$ for every $i = 1, \dots, \ell$. Then, S includes the union of $\{v\}$ and $S_1, \dots, S_\ell$. Let us show the converse inclusion. Choose an arbitrary vertex $x \in S$. If $x = v$, then x is certainly included in the union of $\{v\}$ and $S_1, \dots, S_\ell$. Otherwise, we have $x \in V(G(K)) \setminus K$. Since $V(G(K)) = K \cup \bigcup_{i=1}^{\ell} V(G(K_i))$, the vertex x belongs to S_i for some $i = 1, \dots, \ell$. Therefore, S is included in the union of $\{v\}$ and $S_1, \dots, S_\ell$. Now, we need to show that for every $i = 1, \dots, \ell$ the set S_i satisfies the property required in the lemma. Fix $i = 1, \dots, \ell$. If v belongs to K_i, then S_i belongs to $\mathcal{IS}(G(K_i), v)$ since v also belongs to S. If $v \notin K_i$, then S_i belongs to $\overline{\mathcal{IS}}(G(K_i), K_i \cap K)$ since v is adjacent to any vertex of $K_i \cap K$. Thus the required property is satisfied. This completes the proof of the only-if part.

Next, we prove the if part. Assume that S is the union of $\{v\}$ and $S_1, \dots, S_\ell$ satisfying that $S_i \in \mathcal{IS}(G(K_i), v)$ if $v \in K_i$, and $S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)$ otherwise. When $v \in K_i$, since v is adjacent to all vertices of $K \setminus \{v\}$, every vertex in $S_i \setminus \{v\}$ belongs to $V(G(K_i)) \setminus K$. When $v \notin K_i$, by the definition of $\mathcal{IS}(G(K_i), K \cap K_i)$, every vertex in $S_i \setminus \{v\}$ belongs to $V(G(K_i)) \setminus K$. Therefore, for each $i = 1, \dots, \ell$ it holds that $S_i \setminus \{v\} \subseteq V(G(K_i)) \setminus K$. This implies that $S \setminus \{v\} \subseteq V(G(K)) \setminus K$. Now, we show that for every $i, j \in \{1, \dots, \ell\}$ with $i \neq j$, $(S_i \setminus \{v\}) \cup (S_j \setminus \{v\})$ is independent. To show that, suppose not. Since S_i and S_j are independent, there must be an edge $\{x, y\} \in E$ such that $x \in S_i \setminus \{v\}$ and $y \in S_j \setminus \{v\}$. Since $\{x, y\}$ is an edge of G, it is included in some maximal clique K_{xy} of G. Since T_x and T_y are subtrees of T, this implies that x or y must belong to K. Without loss of generality, assume that x belongs to K. (Remember that $x \in S_i \setminus \{v\}$.) If $S_i \in \mathcal{IS}(G(K_i), v)$, then $S_i \cap K \supseteq \{v, x\}$. This is a contradiction to S_i being independent. If $S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)$, then S_i cannot contain any vertex of K, particularly x. This is also a contradiction. Thus the claim is verified, and it implies that $S \setminus \{v\}$ is an independent set of $G(K)$. Together with the observation that no vertex of $G(K_i) \setminus K$ is adjacent to v if $v \notin K_i$, this further implies that S is an independent set of $G(K)$. Since $v \in S$, this shows that $S \in \mathcal{IS}(G(K), v)$.

To show the uniqueness, suppose that S is the union of $\{v\}, S_1, \dots, S_\ell$ and also the union of $\{v\}, S'_1, \dots, S'_\ell$ such that there exists i with $S_i \neq S'_i$. Without loss of generality assume that $S_i \neq \emptyset$. Choose a vertex $u \in S_i \setminus S'_i$, where $u \neq v$. Then, there must exist $j \neq i$ with $u \in S'_j$. Hence, there exists a node $L \in T(K_i)$ such that $u \in L$ and a node $L' \in T(K_j)$ such that $u \in L'$. Then, by Property (C) in the definition of a clique tree, the nodes on the path connecting L and L' in T contain u. In particular we have $u \in K$. Therefore, u and v belong to the clique K and at the same time they belong to the independent set S. This is a contradiction. □

By a close inspection of the proof above, we can observe that for every $i, j \in \{1, \dots, \ell\}$, $i \neq j$, 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 as above, we obtain the following lemma.

Lemma 3.2. *Let G be a chordal graph and T be a clique tree of G. Choose a maximal clique K of G, and let $K_1, \dots, K_\ell$ be the children in $\text{CHD}(K)$.*

1. *We have $S \in \overline{\mathcal{IS}}(G(K), K)$ if and only if S is the union of $S_1, \dots, S_\ell$ such that $S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)$. Furthermore, such a representation is unique.*
2. *For each $i = 1, \dots, \ell$, 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.*

Proof. (1) Similar to Lemma 3.1, we omit.

(2) First, assume that $S_i \in \mathcal{IS}(G(K_i), v)$ for some $v \in K_i \setminus K$. Since K_i is a clique, S_i cannot include any vertex of $K_i \setminus \{v\}$, particularly of $K \cap K_i$. Therefore, $S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)$. Secondly, assume that $S_i \in \overline{\mathcal{IS}}(G(K_i), K_i)$. Then, S_i includes no vertex of $K_i \cap K$, since $K_i \cap K \subseteq K_i$. Hence, $S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)$. This proves the if part.

Let us prove the only-if part and the uniqueness. Assume that S_i belongs to $\overline{\mathcal{IS}}(G(K_i), K_i \cap K)$. When S_i includes a vertex v of $K_i \setminus K$, we have $S_i \in \mathcal{IS}(G(K_i), v)$. Note that v is a unique element in $S_i \cap (K_i \setminus K)$ since S_i is an independent set and $K_i \setminus K$ is a clique. Therefore, $S_i \notin \mathcal{IS}(G(K_i), u)$ for $u \in (K_i \setminus K) \setminus \{v\}$. When S_i includes no vertex of $K_i \setminus K$, it follows that $S_i \in \overline{\mathcal{IS}}(G(K_i), K_i)$. $\square$

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

$$\begin{aligned}
\mathcal{IS}(G(K)) &= \overline{\mathcal{IS}}(G(K), K) \,\dot\cup \dot{\bigcup}_{v\in K} \mathcal{IS}(G(K), v);\\
\mathcal{IS}(G(K), v) &= \{S \cup \{v\} \mid S = \bigcup_{i=1}^{\ell} S_i, S_i \in \left\{ \begin{array}{ll} \mathcal{IS}(G(K_i), v) & \text{if } v \in K_i \\ \overline{\mathcal{IS}}(G(K_i), K_i \cap K) & \text{otherwise} \end{array} \right\}\};\\
\overline{\mathcal{IS}}(G(K), K) &= \{S \mid S = \bigcup_{i=1}^{\ell} S_i, S_i \in \overline{\mathcal{IS}}(G(K_i), K_i \cap K)\};\\
\overline{\mathcal{IS}}(G(K_i), K_i \cap K) &= \overline{\mathcal{IS}}(G(K_i), K_i) \,\dot\cup \dot{\bigcup}_{u\in K_i\setminus K} \mathcal{IS}(G(K_i), u) \text{ for each } i = 1, \ldots, \ell.
\end{aligned}$$

These equations lead us to the following algorithm to count the number of independent sets in a chordal graph (we remind that an empty set is an independent set).

Algorithm #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;
Output: The number of independent sets in G(K);
if K is a leaf of T **then**
    $\left|\overline{\mathcal{IS}}(G(K), K)\right| := 1$ 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 $\prod_{K'\in \mathbf{CHD}(K)} \left|\overline{\mathcal{IS}}(G(K'), K' \cap K)\right|$;
    foreach $v \in K$ **do** compute $|\mathcal{IS}(G(K), v)|$ by $\left|\overline{\mathcal{IS}}(G(K), K)\right| \times \frac{\prod_{K'\in \mathbf{CHD}(K), v\in K'} |\mathcal{IS}(G(K'), v)|}{\prod_{K'\in \mathbf{CHD}(K), v\in K'} |\overline{\mathcal{IS}}(G(K'), K'\cap K)|}$;
    // The correctness of the equation is proved in the text.
endif.

Theorem 3.3. *The algorithm* #IndSets *outputs the number of independent sets in a chordal graph* $G = (V, E)$ *in* $O(|V| + |E|)$ *time.*

Proof. From Equations 1, we only need to check that Step 10 computes correctly. This can be seen as follows:

$$\begin{aligned}
|\mathcal{IS}(G(K),v)| &= \prod_{K' \in \mathrm{CHD}(K)} \left|\overline{\mathcal{IS}}(G(K'), K' \cap K)\right| \\
&= \prod_{K' \in \mathrm{CHD}(K), v \in K'} |\mathcal{IS}(G(K'),v)| \times \prod_{K' \in \mathrm{CHD}(K), v \notin K'} \left|\overline{\mathcal{IS}}(G(K'), K' \cap K)\right| \\
&= \left|\overline{\mathcal{IS}}(G(K), K)\right| \times \frac{\prod_{K' \in \mathrm{CHD}(K), v \in K'} |\mathcal{IS}(G(K'),v)|}{\prod_{K' \in \mathrm{CHD}(K), v \in K'} \left|\overline{\mathcal{IS}}(G(K'), K' \cap K)\right|}.
\end{aligned}$$

Let us consider the computation time $t(K)$ taken by a call to #IndSetsIter(K). The overall running time of #IndSets is $t(K_r)+O(|K_r|)$. Steps 7 and 8 take $O(t(K'))$ and $O(|K'|)$ time for each $K' \in \mathrm{CHD}(K)$ respectively. Step 9 can be done in $O(\mathrm{CHD}(K))$. Next, we analyze the computation time for Step 10. Since $|\mathcal{IS}(G(K),v)|$ can be computed in $O(|\{K' \in \mathrm{CHD}(K) \mid v \in K'\}|)$ time for each $v \in K$, Step 10 can be done in $O(\sum_{v \in K} |\{K' \in \mathrm{CHD}(K) \mid v \in K'\}|)$ time. Therefore, the accumulated time taken by a call to #IndSetsIter(K_r) is $\sum_{K' \in \mathrm{CHD}(K_r)} (O(t(K')) + O(|K'|)) + O(|\mathrm{CHD}(K_r)|) + O(\sum_{v \in K_r} |\{K' \in \mathrm{CHD}(K_r) \mid v \in K'\}|)$. By expanding $t(K')$ inside the sum, we can see that this is at most $O(\sum_{K \in \mathcal{K}} (|K| + \sum_{v \in K} |\{K' \in \mathrm{CHD}(K) \mid v \in K'\}|))$, where $\mathcal{K}$ denotes the set of nodes in the clique tree, i.e., the family of maximal cliques of G. By Lemma 2.1, we have $\sum_{K \in \mathcal{K}} |K| = O(|V| + |E|)$. Furthermore, it follows that $\sum_{K \in \mathcal{K}} \sum_{v \in K} |\{K' \in \mathrm{CHD}(K) \mid v \in K'\}| = \sum_{v \in V} |\{K' \in \mathcal{K} \mid v \in K'\}| = \sum_{K \in \mathcal{K}} |K| = O(|V| + |E|)$ again by Lemma 2.1. Hence, the overall running time is $O(|V| + |E|)$. □

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 $\mathcal{S}$, argmax$(\mathcal{S})$ denotes the family of sets in $\mathcal{S}$ of the maximum size. 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., $\overline{\mathcal{MIS}}(G,U) := \{S \in \mathcal{MIS}(G) \mid S \cap U = \emptyset\}$. We note that $\mathcal{MIS}(G,v)$ and $\overline{\mathcal{MIS}}(G,U)$ are $\emptyset$ when there is no maximum independent set that satisfies the conditions.

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

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

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

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 let M be the size of a maximum independent set in $\overline{\mathcal{MIS}}(G(K),K) \cup \bigcup_{v \in K} \mathcal{MIS}(G(K),v)$;
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 4.1. *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.

$$\begin{aligned}
\mathcal{IS}(G(K); k) &= \overline{\mathcal{IS}}(G(K), K; k) \,\dot\cup \dot{\bigcup_{v \in K}} \mathcal{IS}(G(K), v; k);\\
\mathcal{IS}(G(K), v; k) &= \{S \mid S = \bigcup_{i=1}^{\ell} S_i, |S| = k, S_i \in \left\{ \begin{array}{ll} \mathcal{IS}(G(K_i), v) & \text{if } v \in K_i \\ \overline{\mathcal{IS}}(G(K_i), K_i \cap K) & \text{otherwise} \end{array} \right\}, v \in S\};\\
\overline{\mathcal{IS}}(G(K), K; k) &= \{S \mid S = \bigcup_{i=1}^{\ell} S_i, |S| = k, S_i \in \overline{\mathcal{IS}}(G(K_i), K_i \cap K)\};\\
\overline{\mathcal{IS}}(G(K_i), K_i \cap K; k) &= \overline{\mathcal{IS}}(G(K_i), K_i; k) \,\dot\cup \dot{\bigcup_{u \in K_i \setminus K}} \mathcal{IS}(G(K_i), u; k).
\end{aligned}$$

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 little more sophisticated algorithm.

Theorem 5.1. *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.*

Proof. Here we show an efficient algorithm that computes $\left|\overline{\mathcal{IS}}(G(K), K; k)\right|$ and $|\mathcal{IS}(G(K), v; k)|$. Fix an arbitrary vertex $v \in K$.

For each $\ell' \le \ell$, we define $\overline{\mathcal{IS}}(G(K), K; k)_{\le \ell'} := \{S \mid S = \cup_{i=1}^{\ell'} S_i, |S| = k, S_i \in \overline{\mathcal{IS}}(G(K_i), K_i \cap K)\}$. Then we can compute $\left|\overline{\mathcal{IS}}(G(K), K; k)\right| = \left|\overline{\mathcal{IS}}(G(K), K; k)_{\le \ell}\right|$ based on the following recursive equation:

$$\left|\overline{\mathcal{IS}}(G(K), K; k)_{\le \ell'}\right| = \left\{ \begin{array}{ll} \left|\overline{\mathcal{IS}}(G(K_1), K_1 \cap K; k)\right| & \text{if } \ell' = 1, \\ \sum_{h=0}^{k} \left(\left|\overline{\mathcal{IS}}(G(K), K; h)_{\le \ell'-1}\right| \times \left|\overline{\mathcal{IS}}(G(K_{\ell'}), K_{\ell'} \cap K; k-h)\right|\right) & \text{otherwise.} \end{array} \right.$$

Hence for a fixed k and each $\ell' = 1, 2, \ldots, \ell$, we can compute $\left|\overline{\mathcal{IS}}(G(K), K; k)_{\le \ell'}\right|$ in $O(k\ell) = O(k|\text{CHD}(K)|)$ time. Simultaneously, we can compute $\left|\overline{\mathcal{IS}}(G(K), K; k')_{\le \ell'}\right|$ for all $0 \le k' \le k$ in $O(k^2\ell)$ time, which will be required in a recursion.

Next we turn to the computation of $\mathcal{IS}(G(K), v; k)$. Then, according to a fixed v, the children of K are divided into two sets such that $K_1, \ldots, K_p$ include v and $K_{p+1}, \ldots, K_\ell$ do not. Here we define two sets as follows.

$$\mathcal{IS}(G(K), v; k)_{\le \ell'} := \{S \mid S = \bigcup_{i=1}^{\ell'} S_i, |S| = k, v \in S, S_i \in \mathcal{IS}(G(K_i), v)\}$$

for each ℓ' with $1 \le \ell' \le p$, and

$$\overline{\mathcal{IS}}(G(K), v; k)_{>\ell''} \quad := \quad \{S \mid S = \bigcup_{i=\ell''+1}^{\ell} S_i, |S| = k, S_i \in \overline{\mathcal{IS}}(G(K_i), K_i \cap K)\}$$

for each ℓ'' with $p \le \ell'' \le \ell - 1$. We note that each S in $\mathcal{IS}(G(K), v; k)_{\le \ell'}$ contains v, and each S in $\overline{\mathcal{IS}}(G(K), v; k)_{>\ell''}$ does not. Then, it holds that

$$|\mathrm{IS}(G(K), v; k)| = \sum_{h=0}^{k} \left(|\mathcal{IS}(G(K), v; h)_{\le p}| \times \left|\overline{\mathcal{IS}}(G(K), v; k-h)_{>p}\right| \right).$$

Using the same technique above, we can compute $|\mathcal{IS}(G(K), v; h)_{\le p}|$ from $h = 0$ up to $h = k$ in $O(hp)$ time in total, and $\left|\overline{\mathcal{IS}}(G(K), v; h')_{>p}\right|$ from $h' = k$ down to $h' = 0$ in $O(h'(\ell - p))$ time in total. Thus we can obtian $|\mathrm{IS}(G(K), v; k)|$ for a fixed v and k in $O(hp + h'(\ell - p)) = O(k\ell)$ time. Simultaneously, for a fixed v, we can compute $|\mathrm{IS}(G(K), v; k')|$ for all $0 \le k' \le k$ in $O(k^2\ell) = O(k^2|\textsc{chd}(K)|)$ time.

We further reduce the computation time. At a clique K with children $K_1, \ldots, K_\ell$, we first compute $\left|\overline{\mathcal{IS}}(G(K), K; k')\right|$ with $0 \le k' \le k$ in $O(k^2\ell)$ time. Next, for all $v \in K$ and $k' = 0, \ldots, k$, we compute $|\mathcal{IS}(G(K), v; k')|$. For a fixed v, we can compute $|\mathcal{IS}(G(K), v; k')|$ for all $0 \le k' \le k$ in $O(k^2\ell)$ time. When we compute $|\mathcal{IS}(G(K), v; k')|$ for all $v \in K$, we can omit some computation for $\overline{\mathcal{IS}}(G(K), v; k)_{>\ell''} = \{S \mid S = \bigcup_{i=\ell''+1}^{\ell} S_i, |S| = k, S_i \in \overline{\mathcal{IS}}(G(K_i), K_i \cap K)\}$ since it is independent from v. More precisely, $\left|\{S_i \in \overline{\mathcal{IS}}(G(K_i), K_i \cap K) \mid |S_i| = k'\}\right|$ for each $k' \le k$ can be precomputed in $O(k^2)$ time in total. Hence, we can compute $\left|\overline{\mathcal{IS}}(G(K), K; k')\right|$ and $|\mathcal{IS}(G(K), v; k')|$ for all $v \in K$ and $k' = 0, \ldots, k$ in $O(k^2(\ell + \sum_{v \in K} |\{K' \in \textsc{chd}(K) \mid v \in K'\}|))$ time. Therefore, the total computation time over all iterations can be bounded in the same way as the above section, and we have the theorem. □

6 Enumeration

In this section we give enumeration algorithms using the same technique as our counting algorithms in the previous sections.

First, we describe a simple algorithm to enumerate all independent sets in a chordal graph. Equations 1 in Section 3 give a recursive structure for the family of independent sets. Thus we can construct the following algorithm in a straightforward way. We first set $S := \emptyset$. Then, for each maximal clique K of a given chordal graph, we iteratively add a vertex of $K \setminus \textsc{prt}(K)$ into S (or no vertex to S) in a depth-first-search manner. Then each vertex in $K \setminus \textsc{prt}(K)$ gives us a distinct independent set. Hence we pick up each of them to enumerate all independent sets. A simple implementation of the algorithm is as follows (for notational convenience, let $K_0 := \emptyset$ and $K_{n+1} := \emptyset$).

Algorithm EnumIndSets
Input: A chordal graph $G = (V, E)$;
Output: All independent sets in G;
1: construct a rooted clique tree T of G;
2: let $K_1, \ldots, K_n$ be the maximal cliques ordered in a depth first manner on T;
3: set $S := \emptyset$ and **call** EnumIndSetsIter(K_0, S).

Procedure EnumIndSetsIter(K_i, S)
Input: A maximal clique K_i and an independent set S;
Output: All independent sets S' such that $S' \cap (K_1 \cup K_2 \cup \cdots \cup K_i) = S$;
4: **if** $i = n + 1$ **then** // output an independent set at the bottom level
5: **output** S and **return**;
6: **else**
7: **call** EnumIndSetsIter(K_{i+1}, S);
8: **if** $K_i \cap S = \emptyset$ **then** // S includes no vertex of K_i

9: **foreach** $u \in K_i \setminus \textsc{prt}(K_i)$ **do call** $\mathsf{EnumIndSetsIter}(K_{i+1}, S \cup \{u\})$;
10: **endif**
11: **endif.**

The correctness of the simple algorithm follows from Equations 1 in Section 3. Since G is a chordal graph, the number n of maximal cliques is bounded by $|V|$. Hence the algorithm outputs each independent set in $O(|V|)$ time. More precisely, the algorithm consumes $O(|V|)$ time between two consecutive independent sets. We modify the simple algorithm to reduce the time complexity.

Theorem 6.1. *After* $O(|V|(|V| + |E|))$ *time and* $O(|V|(|V| + |E|))$ *space precomputation, all independent sets in a chordal graph can be enumerated in a (worst-case) constant time for each.*

We remind that the number of independent sets can be exponential, which implies that the cost of a polynomial time precomputation can be negligible.

Proof. Let $\mathcal{T}$ be a computation tree of the simple algorithm, in which each node (K, S) corresponds to a recursive call to $\mathsf{EnumIndSetsIter}(K, S)$ generated by the algorithm[5]. A node (K, S) is the parent of a node (K', S') if $\mathsf{EnumIndSetsIter}(K', S')$ is invoked in $\mathsf{EnumIndSetsIter}(K, S)$ (or $\mathsf{EnumIndSets}$ if $K' = K_1$ and $S' = \emptyset$). When $K = K_{n+1}$, each node (K, S) is a leaf and the algorithm outputs an independent set.

A node (K, S) is called *unnecessary* if it has exactly one child in $\mathcal{T}$. By lines 7, 8, and 9 in the algorithm, a node (K, S) is unnecessary if and only if $K \cap S \neq \emptyset$. We also call a node (K, S) *necessary* if it is not unnecessary. In general, $\mathcal{T}$ may contain many unnecessary nodes, and $\mathcal{T}$ cannot be traversed by the algorithm efficiently. Hence we here aim at skipping unnecessary nodes of $\mathcal{T}$ in the computation. Let $\mathcal{T}'$ be the reduced computation tree, which only contains necessary nodes. We say that a vertex $v \in V$ *hits* a clique K if $v \in K$.

At a necessary node (K_i, S), the algorithm picks up each vertex u in $K_i \setminus \textsc{prt}(K_i)$. Then, since S contains no vertex in K_i and $u \in K_i$, the next necessary node(s) visited by the simple algorithm after K_i depends on K_i and u as we describe below.

First, we assume that u does not hit some cliques which are descendants of K_i in the rooted clique tree T. Let $K_{j_0}, K_{j_1}, \cdots, K_{j_\ell}$ be the descendant cliques of K_i that are the roots of the subtrees obtained by removing the maximal cliques hit by u from the rooted clique tree T. We assume that $(i <)\ j_0 < j_1 < \cdots < j_\ell$. Then those roots are the necessary nodes with respect to K_i and u, and it suffices to visit them after the node (K_i, S) in the reduced computation tree $\mathcal{T}'$ as children of the nodes (K_i, S) (with the independent set $S \cup \{u\}$). Thus we define $\textsc{next}(K_i, u)$ by the set $\{K_{j_0}, K_{j_1}, \cdots, K_{j_\ell}\}$ and we implement $\textsc{next}(K_i, u)$ by a linked list.

Second, we assume that u hits all cliques that are descendants of K_i in T. Then we define $\textsc{next}(K_i, u)$ by $\emptyset$ unless u hits the last clique K_n. When u hits K_n, we define $\textsc{next}(K_i, u) = \{K_{n+1}\}$ to jump to step 5.

The modified algorithm performs the following step 9' instead of the step 9:

9': foreach $u \in K_i \setminus \textsc{prt}(K_i)$ **do**
foreach $K \in \textsc{next}(K_i, u)$ **do call** $\mathsf{EnumIndSetsIter}(K, S \cup \{u\})$;

By the above arguments, the modified algorithm correctly performs its computation along the computation tree $\mathcal{T}'$. We now show its complexity. Since $S \cup \{u\}$ is an independent set and $u \in K_i$, the set $\textsc{next}(K_i, u)$ is uniquely determined by u and i; it consists of the nodes K_j of the rooted clique tree T such that $u \notin K_j$, $j > i$, and all maximal cliques K' between K_i and K_j on T contain u. Since $u \in K_i$ and $n = O(|V|)$, the number of pairs (K_i, u) is $O(|V|+|E|)$. For each pair (K_i, u) with $u \in K_i$, the set $\textsc{next}(K_i, u)$ consists of $O(n)$ cliques. Hence $\textsc{next}(K_i, u)$ can be computed in $O(n)$ time by a simple depth first search on T. Therefore, all the $\textsc{next}(K_i, u)$ can be precomputed in $O(n(|V| + |E|)) = O(|V|(|V| + |E|))$ time and space. Since $\textsc{next}(K_i, u)$ is a linked list for each K_i and u, the algorithm can obtain each $K \in \textsc{next}(K_i, u)$ in $O(1)$ time in step 9'.

Now we finalize the proof. Every inner node of $\mathcal{T}'$ has at least two children. Thus the total number of the inner nodes is bounded by the number of leaves, which is equal to the number of independent sets. Therefore, the total number of the nodes in $\mathcal{T}'$ is $O(M)$, where M is the number of independent sets. Each traverse of an edge of the computation tree $\mathcal{T}'$ takes $O(1)$ time. Using the odd-even search technique (each output is controlled by the parity

[5]To distinguish a vertex in G, we say $\mathcal{T}$ consists of "nodes."

of the depth of the node in $\mathcal{T}'$; see, e.g., [17]) to make the output interval balanced, all independent sets can be enumerated in a constant time for each. $\square$

Corollary 6.2. *(1) After* $O(|V|^2(|V|+|E|))$ *time and* $O(|V|^2(|V|+|E|))$ *space precomputation, all maximum independent sets in a chordal graph can be enumerated in a constant time for each. (2) After* $O(k|V|(|V|+|E|))$ *time and* $O(k|V|(|V|+|E|))$ *space precomputation, all independent sets of size* k *in a chordal graph can be enumerated in a constant time for each.*

Proof. Let T be a rooted clique tree of a chordal graph G defined by the maximal cliques $K_1, K_2, \dots, K_n$. Then the simple implementations of the algorithms from Equations 2 and 3 are straightforward. In the algorithms, we handle the size k' of an independent set as follows. For given maximal cliques, we can precompute the size of a maximum independent set in the (chordal) graph $G(K_i)$ induced by the subtree rooted at K_i. Using the information, we can define and precompute a list $\textsc{next}(K_i, u; k')$ of the next necessary maximal cliques K with respect to K_i and u such that $G(K)$ can provide an independent set of size k'. Then, we have to consider the case that step 7 of Algorithm EnumIndSetsIter(K_{i+1}, S) is skipped since S and K_{i+1} do not have enough vertices to make an independent set of size k'. More precisely, at node (K_i, S), the algorithm (pre)determine if $K_{i+1}, \dots, K_n$ has enough size to produce an independent set of size k'. If the algorithm cannot make an independent set of required size k' without adding one vertex from K_i, it skips step 7 at node (K_i, S). In the case, if $|K_i \setminus \textsc{prt}(K_i)| = 1$, the node (K_i, S) has one child in the computation tree, that is, the node (K_i, S) becomes unnecessary. Thus we have to add nodes (K_i, S) with the conditions (one vertex has to be added from $K_i \setminus \textsc{prt}(K_i)$, and $|K_i \setminus \textsc{prt}(K_i)| = 1$) to unnecessary nodes. Moreover, in the case, the difference between two consecutive outputs (or independent sets) is not constant in general. Hence we have to design a code for such outputs, which can be done in a standard technique. The modification of the algorithms using the notion $\textsc{next}(K_i, u; k')$ is straightforward and tedious, so omitted here. $\square$

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. First we consider the following counting problem.

Problem: # MAXIMAL INDEPENDENT SETS IN A CHORDAL GRAPH
Instance: A chordal graph $G = (V, E)$;
Output: The number of maximal independent sets of G.

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 7.1. *The problem "# 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 $\bigcup \mathcal{F} = X$. The following problem is #P-complete [18].

Problem: # SET COVERS
Instance: A finite set X and a family $\mathcal{S} \subseteq 2^X$;
Output: The number of set covers of X.

Proof. Proof of Theorem 7.1. The membership in #P of "# MAXIMAL INDEPENDENT SETS IN A CHORDAL GRAPH" is immediate. To show the #P-hardness, we reduce "# SET COVERS" to "# MAXIMAL INDEPENDENT SETS IN A CHORDAL GRAPH" in polynomial time.

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 # SET COVERS. Let us put $\mathcal{S} := \{S_1, \dots, S_t\}$. 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 $\mathcal{S}' := \{S'_1, \dots, S'_t\}$. 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,

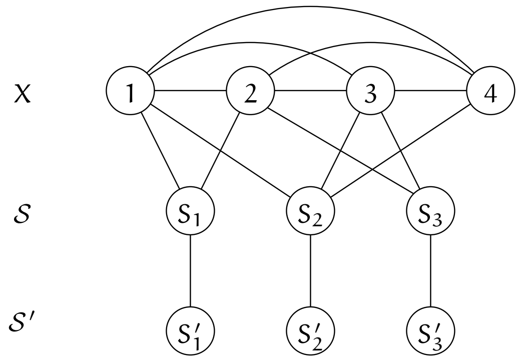

Figure 1: Illustration of the reduction. In this example, $X = \{1, 2, 3, 4\}$, $\mathcal{S} = \{S_1, S_2, S_3\}$, $S_1 = \{1, 2\}$, $S_2 = \{1, 3, 4\}$, and $S_3 = \{2, 3\}$.

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. This construction can be done in polynomial time. Figure 1 illustrates the construction.

First, let us check that the constructed graph G is indeed chordal. Let C be a cycle of length at least four in G. Since the degree of a vertex in $\mathcal{S}'$ is one, they do not take part in any cycle of G. So forget them. Since $\mathcal{S}$ is an independent set of G, vertices in $\mathcal{S}$ cannot appear along C in a consecutive manner. Then, since the length of C is at least four, there have to be at least two vertices of X which appear in C not consecutively. Then, these two vertices give a chord since X is a clique of G. Hence, G is 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 $x \in X$. Since X is a clique of G, U cannot contain any other vertices of X. Let $G_x := G \setminus N_G[x]$. (Remember that $N_G[x]$ is the closed neighborhood of x, i.e., the set of vertices adjacent to x in G and x itself.) 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, a vertex $S' \in \mathcal{S}'$ such that $x \in S$ is an isolated vertex of G_x. Therefore, this vertex must belong to U by the maximality of U. For each $S \in \mathcal{S}$ such that $x \notin S$, U must contain either S or S', but not both. This means that the number of maximal independent sets containing x is exactly $2^{|\{S \in \mathcal{S} \mid x \notin S\}|}$.

Case 2. Consider the case in which U contains no vertex of X. Then, for each $S \in \mathcal{S}$, due to the maximality, U must contain either S or S'. Furthermore, $U \cap \mathcal{S}$ has to be a set cover of X (otherwise an element of X not covered by $U \cap \mathcal{S}$ could be included in U). Hence, 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 $\sum_{x \in X} 2^{|\{S \in \mathcal{S} \mid x \notin S\}|}$. Since the last sum can be computed in polynomial time, this concludes the reduction. □

As a variation, let us consider the following problem.

Problem: # MINIMUM MAXIMAL INDEPENDENT SETS IN A CHORDAL GRAPH	
Instance: A chordal graph $G = (V, E)$;	
Output: The number of minimum maximal independent sets of G.	

Note that a minimum maximal independent set in a chordal graph can be found in polynomial time [9]. In contrast to that, it is hard to count the number of minimum maximal independent sets in a chordal graph:

Theorem 7.2. *The problem "*# MINIMUM MAXIMAL INDEPENDENT SETS IN A CHORDAL GRAPH*" is* #P*-complete.*

Proof. We use the same reduction as in the proof of Theorem 7.1. Look at the case distinction in that proof again. The maximal independent sets arising from Case 1 have $|\mathcal{S}| + 1$ elements, while the maximal independent sets from Case 2 have $|\mathcal{S}|$ elements. Therefore, the minimum maximal independent sets of the graph G constructed in that proof are exactly the maximal independent sets arising from Case 2, which precisely correspond to the set covers of X. $\square$

We note that the chordal graph G in this section is very close to a *split graph* G' which consists of the clique X and an independent set S in G. However, for a split graph, it is easy to solve the problems of this section in polynomial time since a split graph contains only two types of maximal independent sets; one type consists of one vertex v in X and all vertices in $S \setminus N(v)$, and the other possible one is S itself.

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.

Problem: MINIMUM WEIGHTED MAXIMAL INDEPENDENT SET IN A CHORDAL GRAPH
Instance: A chordal graph $G = (V, E)$ and a vertex weight $w : V \to \mathbb{N}$;
Output: A minimum weighted maximal independent set of G.

Here, the weight of a vertex subset is the sum of the weights of its vertices.

Notice that there is a linear-time algorithm when the weight of each vertex is zero or one [9]. On the contrary, we show that the problem is actually hard when the weight is arbitrary.

Theorem 8.1. *The problem "*MINIMUM WEIGHTED MAXIMAL INDEPENDENT SETS 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.

Problem: MINIMUM SET COVER
Instance: A finite set X and a family $\mathcal{S} \subseteq 2^X$;
Output: A minimum-size set cover of X.

It is known that MINIMUM SET COVER is NP-hard.

Proof. Proof of Theorem 8.1. For a given instance of MINIMUM SET COVER, we use the same construction of a graph G as in the proof of Theorem 7.1. 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 7.1, 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. $\square$

We can further show the hardness to get an approximation algorithm running in polynomial time. The precise statement is as follows.

Theorem 8.2. *There is no polynomial-time algorithm for* MINIMUM MAXIMAL WEIGHTED INDEPENDENT SET IN A CHORDAL GRAPH *with approximation ratio* $c \ln |V|$*, for some fixed constant* c*, unless* $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$.

Note that $\mathsf{DTIME}(t)$ is the class of languages which have a deterministic algorithm running in time t.

It was shown by Feige [10] that there is no polynomial-time algorithm for MINIMUM SET COVER with approximation ratio $c' \ln |V|$, for any fixed constant $c' \leq 1$, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$. This holds even if the size of the family $\mathcal{S}$ is bounded by a polynomial $p(|X|)$ of $|X|$.

Now we are ready to prove Theorem 8.2.

Proof. Proof of Theorem 8.2. Suppose that there exists a polynomial-time algorithm A with approximation ratio $c \ln |V|$ for MINIMUM WEIGHTED MAXIMAL INDEPENDENT SET IN A CHORDAL GRAPH. (The constant c will be determined later.) We use the algorithm A to get a polynomial-time algorithm with approximation ratio $c' \ln |X|$ for MINIMUM SET COVER. Then, this will imply that $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$.

Let X be a finite set and $\mathcal{S} \subseteq 2^X$ be a family of subsets of X. We assume that $|X| \geq 3$ and $1 \leq |\mathcal{S}| \leq |X|^d$ for some natural number d. From them, we construct a graph G exactly in the same way as in the proof of Theorem 7.1. Setting $\alpha := \lceil c \ln(2|X|^d)) \rceil |\mathcal{S}|$, we define a weight w as follows: $w(x) := 2\alpha^2 + 1$ for every $x \in X$; $w(S) := 2\alpha$ for every $S \in \mathcal{S}$; $w(S') := 1$ for every $S' \in \mathcal{S}'$. This is our construction. (Note that this construction can be done in polynomial time.)

Denote by OPT an arbitrary (fixed) minimum weighted maximal independent set of G, by APX an output of the algorithm A for G, and by $w(\mathrm{OPT})$ and $w(\mathrm{APX})$ the weights of them. Since the number of vertices in G is $|X| + 2|\mathcal{S}|$, which is at most $|X| + 2|X|^d \leq 3|X|^d$, it follows that $w(\mathrm{APX}) \leq c \ln(3|X|^d) w(\mathrm{OPT})$.

As in the proof of Theorem 8.1, $\mathcal{S}$ is a maximal independent set of G and its weight is $2\alpha|\mathcal{S}|$. Therefore, it holds that $w(\mathrm{OPT}) \leq 2\alpha|\mathcal{S}|$.

Now, suppose that there exists an element $x \in X$ which is contained in APX. Then, $w(\mathrm{APX}) \geq w(x) = 2\alpha^2 + 1$. This implies that $2\alpha^2 < w(\mathrm{APX}) \leq c \ln(3|X|^d) w(\mathrm{OPT}) \leq \lceil c \ln(3|X|^d) \rceil \times 2\alpha|\mathcal{S}| = 2\alpha^2$. This is a contradiction. Thus, no element $x \in X$ belongs to APX. This means that $\mathrm{APX} \cap \mathcal{S}$ is a set cover of X. Let $\mathcal{C} := \mathrm{APX} \cap \mathcal{S}$ and we show that $\mathcal{C}$ approximates the optimal value for MINIMUM SET COVER within a factor of $c' \ln |X|$.

Again, by the same argument as in the proof of Theorem 8.1, we get $w(\mathrm{APX}) = (2\alpha - 1)|\mathcal{C}| + |\mathcal{S}|$. Let $\mathcal{C}^*$ be a minimum set cover of X. Then, similarly we get $w(\mathrm{OPT}) = (2\alpha - 1)|\mathcal{C}^*| + |\mathcal{S}|$. Since $w(\mathrm{APX}) \leq c \ln(3|X|^d) w(\mathrm{OPT})$, it follows that $(2\alpha - 1)|\mathcal{C}| + |\mathcal{S}| \leq c \ln(3|X|^d)((2\alpha - 1)|\mathcal{C}^*| + |\mathcal{S}|) \leq c \ln(3|X|^d)(2\alpha - 1)|\mathcal{C}^*| + \alpha$. Hence, we obtain

$$
\begin{aligned}
|\mathcal{C}| &\leq c \ln(3|X|^d)|\mathcal{C}^*| + \frac{\alpha - |\mathcal{S}|}{2\alpha - 1} \\
&\leq c \ln(3|X|^d)|\mathcal{C}^*| + \frac{\alpha - \frac{1}{2}}{2\alpha - 1} \\
&= c \ln(3|X|^d)|\mathcal{C}^*| + \frac{1}{2} \\
&\leq c \ln(3|X|^d)|\mathcal{C}^*| + \frac{1}{2} \ln(3|X|^d)|\mathcal{C}^*| \\
&= (c + \frac{1}{2}) \ln(3|X|^d)|\mathcal{C}^*| \\
&\leq (c + \frac{1}{2}) \ln(|X|^{d+1})|\mathcal{C}^*| \\
&= ((d+1)(c + \frac{1}{2}) \ln |X|)|\mathcal{C}^*|.
\end{aligned}
$$

Setting $c = \frac{c'}{d+1} - \frac{1}{2}$ gives approximation ratio $c' \ln |X|$. □

In the proof, we did not aim at optimizing the constant c'.

Note: After this work, we found that Chang proved the NP-completeness of the weighted independent domination problem on a chordal graph [7] which is essentially equivalent to Theorem 8.1. However, we leave Theorem 8.1 with its proof since the reduction in the proof is extended to show Theorem 8.2.

Acknowledgements

The authors thank Masashi Kiyomi for enlightening discussions and pointing out the work by Chang [7]. The authors are grateful to L. Shankar Ram for pointing out a paper [5]. The authors also thank the anonymous referees for their detailed comments and suggestions that improved the presentation significantly.

References

[1] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the Desirability of Acyclic Database Schemes. *Journal 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. *Discrete Mathematics*, 9:205–212, 1974.

[5] L. S. Chandran. A Linear Time Algorithm for Enumerating All the Minimum and Minimal Separators of a Chordal Graph. In *COCOON 2001*, pages 308–317. Lecture Notes in Computer Science 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] G. J. Chang. The Weighted Independent Domination Problem Is NP-Complete for Chordal Graphs. *Discrete Applied Mathematics*, 143:351–352, 2004.

[8] D. Eppstein. All Maximal Independent Sets and Dynamic Dominance for Sparse Graphs. In *Proc. 16th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 451–459. ACM, 2005.

[9] M. Farber. Independent Domination in Chordal Graphs. *Operations Research Letters*, 1(4):134–138, 1982.

[10] U. Feige. A Threshold of $\ln n$ for Approximating Set Cover. *Journal of the ACM*, 45(4):634–652, 1998.

[11] J. Flum and M. Grohe. The Parameterized Complexity of Counting Problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.

[12] D. R. Fulkerson and O. A. Gross. Incidence Matrices and Interval Graphs. *Pacific J. Math.*, 15:835–855, 1965.

[13] F. Gavril. Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

[14] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Annals of Discrete Mathematics 57. Elsevier, 2nd edition, 2004.

[15] P. N. Klein. Efficient Parallel Algorithms for Chordal Graphs. *SIAM Journal on Computing*, 25(4):797–827, 1996.

[16] J. Y.-T. Leung. Fast Algorithms for Generating All Maximal Independent Sets of Interval, Circular-Arc and Chordal Graphs. *Journal of Algorithms*, 5:22–35, 1984.

[17] S. Nakano and T. Uno. Constant Time Generation of Trees with Specified Diameter. In *Graph-Theoretic Concepts in Computer Science (WG 2004)*, pages 33–45. Lecture Notes in Computer Science Vol. 3353, Springer-Verlag, 2005.

[18] J. S. Provan and M. O. Ball. The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected. *SIAM Journal on Computing*, 12:777–788, 1983.

[19] 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.

[20] J. P. Spinrad. *Efficient Graph Representations*. American Mathematical Society, 2003.

[21] 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.

[22] S. P. Vadhan. The Complexity of Counting in Sparse, Regular, and Planar Graphs. *SIAM Journal on Computing*, 31(2):398–427, 2001.