コーダルグラフの独立点集合の数えあげ問題

岡本 吉央[†‡], 宇野 毅明[§], 上原 隆平[¶]

Abstract. コーダルグラフの独立点集合の数えあげや列挙について研究した．まず，以下の結果をえた．(1) 独立点集合の個数を数える線形時間アルゴリズム (2) 最大独立点集合の個数を数える線形時間アルゴリズム (3) 与えられたサイズの独立点集合を数える多項式時間アルゴリズム．また，これらの列挙が，一つあたりのならし計算量 O(1) で可能であることを示した．一方，以下の問題は#P 完全であることを示した．(1) 極大独立点集合の数えあげ問題 (2) 要素数最小の極大独立点集合の数えあげ問題．さらに，重み最小の極大独立点集合を見つける問題が NP 困難で，かつ近似も困難であることを示した．
Keywords: コーダルグラフ, 数えあげ, 列挙, 独立点集合, NP 困難性, #P 困難性, 多項式時間アルゴリズム.

Counting the Independent Sets of a Chordal Graph

Yoshio Okamoto[†‡], Takeaki Uno[§], Ryuhei Uehara[¶]

Abstract. We study some counting and enumeration problems for chordal graphs, especially concerning independent sets. We provide 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. On the contrary, 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 show that enumerations (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, while finding a minimum weight 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].

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, and it is sometimes called a triangulated graph, or a rigid circuit graph. Chordal graphs have numerous applications in, for example, sparse matrix computation (e.g., see Blair & Peyton [2]), relational databases [1], and computational biology [4]. The class of chordal graphs contains the interval graphs, which come along a lot of real world applications including some scheduling and sequencing problems. Recently as a generalization of chordal graphs, the concept of a chordal probe graph is introduced by Golumbic & Lipshteyn [11] along applications in computational biology. Also, the class of chordal graphs forms a subclass of perfect graphs [9, 10].

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 weight clique problem, the maximum weight independent set problem, the minimum coloring problem [8], the minimum maximal independent set problem [5]. There are also parallel algorithms to solve some of these problems efficiently [12]. However, fewer problems have been studied for counting and enumeration. For enumeration in chordal graphs, the only algorithm we are aware of is one due to Fulkerson & Gross [7] which enumerates all maximal cliques in a chordal graph.

[†] Institute of Theoretical Computer Science, Department of Computer Science, ETH Zürich, Switzerland. okamotoy@inf.ethz.ch
[‡] Supported by Berlin/Zurich Graduate Program “Combinatorics, Geometry, and Computation (CGC)” financed by ETH Zurich and the German Science Foundation (DFG).
[§] 国立情報学研究所 (National Institute of Informatics), uno@nii.jp
[¶] 駒澤大学 自然科学教室 (Komazawa University, Natural Science Faculty), uehara@komazawa-u.ac.jp

In this work, we study the problems concerning independent sets in a chordal graph. Especially, we give the following kind of efficient algorithms: (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; (3) a polynomial-time algorithm to count the number of independent sets of a fixed size, in particular running in linear time when the size is constant. The basic idea of these algorithms is to invoke a clique tree associated with a chordal graph and perform the dynamic programming on the clique tree. Since a clique tree can be generated in linear time and the structure of the clique tree is simple, this approach leads to simple and efficient algorithms for the problems above. Along the same idea, we can also enumerate the independent sets, the maximum independent sets, and the independent sets of constant size in a chordal graph in constant amortized time per output.

On the contrary, we show that the following counting problems are #P-complete: (1) counting the number of maximal independent sets in a chordal graph; (2) counting the number of minimum maximal independent sets in a chordal graph. Using a modified version of the same reduction, we furthermore show that the problem to find a minimum weight maximal independent set is NP-hard, and even hard to approximate (more precisely 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 $\mathsf{NP} \subseteq \mathsf{ZTIME}(n^{O(\log \log n)})$).

2 Preliminaries

The *neighborhood* of a vertex v in a graph $G = (V, E)$ is the set $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$, and the *degree* of a vertex v is $|N_G(v)|$ and is denoted by $\deg_G(v)$. For the vertex set 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 index G. We denote the closed neighborhood $N(v) \cup \{v\}$ 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 $F = \{\{u, v\} \mid \{u, v\} \in E \text{ for } u, v \in U\}$, and denoted by $G[U]$. The *complement* of a graph $G = (V, E)$ is defined as $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{\{u, v\} \mid \{u, v\} \notin E\}$. A vertex set I is an *independent set* if $G[I]$ contains no edge, and then the graph $\bar{G}[I]$ is said to be *clique*.

For a given graph $G = (V, E)$, a sequence of the distinct vertices $v_0, v_1, \ldots, v_l$ is a *path*, denoted by $(v_0, v_1, \ldots, v_l)$, if $\{v_j, v_{j+1}\} \in E$ for each $0 \leq j < l$. The *length* of a path $(v_0, v_1, \ldots, v_l)$ is the number l. A *cycle* is a path beginning and ending with the same vertex. 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. Given a graph $G = (V, E)$, a vertex $v \in V$ is *simplicial* in G if $G[N(v)]$ is a clique in G.

An ordering $v_1, \ldots, v_n$ of the vertices of V is a *perfect elimination ordering* of G if the vertex v_i is simplicial in $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ for all $i = 1, \ldots, n$. Then a graph is chordal if and only if it has a perfect elimination ordering (see, e.g., [3, Section 1.2] for further details). Given a chordal graph a perfect elimination ordering of the graph can be found in linear time [15, 17].

It is well known that a graph $G = (V, E)$ is chordal if and only if it is the intersection graph of subtrees of a tree T (see [3, Section 1.2] for further details). For the tree T, it can be assumed that the nodes of T are the maximal cliques of G, and the subtrees T_v, $v \in V$, are defined by the occurrences of the vertex v in the maximal cliques of G. Such a tree T is called a *clique tree* of G. Given a chordal graph $G = (V, E)$, we take any perfect elimination ordering $v_1, v_2, \ldots, v_n$. Let $C(v_i)$ be the clique induced by the set $N[v_i]$ in $G[v_i, v_{i+1}, \ldots, v_n]$. Then, from the perfect elimination ordering, we can construct a clique tree of G in linear time [16]. In the clique tree T, each node corresponds to a maximal clique in G, which can be represented by $C(v)$ for some v in V.

3 Linear-Time Algorithm for Counting the Independent Sets

In this section, we describe our 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 $G(K)$ induced by subtrees of the clique tree. Any two these subtrees share vertices 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, recursively, by using the recursive equations.

First, we introduce some notations, and state some lemmas. Given a chordal graph $G = (V, E)$, we fix a perfect elimination ordering of G, and construct the corresponding clique tree T of G by the algorithm in [16]. We now pick any node in the clique tree T, regard the node as the root of T, and denote it by K_r. For a maximal clique K in G, another maximal clique K' in G is a *descendant* of K if K' is a descendant of K in the clique tree T. For

convenience, we consider K itself a descendant of K as well. Let $\mathrm{PRT}(K)$ be the parent of K. We define $\mathrm{PRT}(K_r)$ by $\emptyset$. We denote by $T(K)$ the subtree of T rooted at the node which corresponds to the maximal clique K. $G(K)$ denotes the graph induced by the vertices included in at least one node in $T(K)$. That is, $G(K)$ is a chordal graph with the clique tree $T(K)$. The following lemma is the key to our counting algorithm.

Lemma 1. *Let G be a chordal graph and T be the clique tree of G. Let K and K' be maximal cliques of G such that K' is not a descendant of K. Then, if a vertex v of $G(K)$ satisfies one of the following two conditions, then v belongs to $K \cap \mathrm{PRT}(K)$. (1) v belongs to K', or (2) v is adjacent to some vertex u of $G(K')$ not included in $G(K)$.*

Proof. Suppose that (1) a vertex v of $G(K)$ belongs to K'. Then, both K' and a descendant H of K are nodes of T_v. Since K' is not a descendant of K, the path connecting H and K' in the clique tree must include both K and $\mathrm{PRT}(K)$. Thus, $v \in K \cap \mathrm{PRT}(K)$.

Suppose that (2) a vertex v of $G(K)$ is adjacent to a vertex u of $G(K')$ outside $G(K)$. Then, T_u includes no descendant of K. Since T_v includes a descendant of K and has a non-empty intersection with T_u, T_v must includes both K and $\mathrm{PRT}(K)$. Thus, $v \in K \cap \mathrm{PRT}(K)$. □

For a graph G, let $\mathcal{IS}(G)$ be the set of independent sets in G. For a vertex v, let $\mathcal{IS}(G, v)$ be the set 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 set 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 2. *Let G be a chordal graph and T be the clique tree of G. Choose a maximal clique K of G, and let $K_1, \ldots, K_l$ be the children of K in T. 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 $S_1, \ldots, S_l$ such that $S_i \in \mathcal{IS}(G(K_i), v)$ if K_i includes v, and $S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)$ otherwise. In particular, such a representation is unique.*

Proof. Suppose that $S \in \mathcal{IS}(G(K), v)$. Let $S_i := S \cap V(G(K_i))$. Then, S includes the union of $\{v\}$ and $S_1, \ldots, S_l$. Since S includes v, S_i belongs to $\mathcal{IS}(G(K_i), v)$ if $v \in K_i$. When $v \notin K_i$, S_i belongs to $\overline{\mathcal{IS}}(G(K_i), K_i \cap K)$ since v is adjacent any vertex of $K_i \cap K$. Since any vertex of $G(K) \setminus \{v\}$ is either adjacent to v or included in $G(K_i)$ for some i, S is equal to the union of $\{v\}$ and $S_1, \ldots, S_l$.

Suppose that S is the union of $\{v\}$ and $S_1, \ldots, S_l$ 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. Since v is adjacent to all vertices of $K \setminus \{v\}$, any vertex in $S_i \setminus \{v\}$ is included in $V(G(K_i)) \setminus K$. Hence, from Lemma 1, any vertices in $S_i \setminus \{v\}$ and $S_j \setminus \{v\}$ are not adjacent to each other if $i \neq j$. Together with the observation that no vertex of $G(K_i) \setminus K$ is adjacent to v if $v \notin K_i$, this implies that S is an independent set of $G(K)$. Since $v \in S$, this shows that $S \in \mathcal{IS}(G(K), v)$.

For the uniqueness, suppose that S is the union of $\{v\}, S_1, \ldots, S_l$ and also the union of $\{v\}, S'_1, \ldots, S'_l$ such that there exists i with $S_i \neq S'_i$. Without loss of generality assume $S_i \neq \emptyset$. Choose a vertex $u \in S_i \setminus S'_i$, where $u \neq v$. Then, there must exist $j \neq i$ with $u \in S'_j$. This means that the nodes on the path connecting S_i and S'_j in T include u; in particular $u \in K$. Namely, 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 similar discussion, we obtain the following lemma.

Lemma 3. *Let G be a chordal graph and T be the clique tree of G. Choose a maximal clique K of G, and let $K_1, \ldots, K_l$ be the children of K in T. If K is a leaf of the clique tree, $l = 0$.*

1. *$S \in \overline{\mathcal{IS}}(G(K), K)$ if and only if S is the union of $S_1, \ldots, S_l$ such that $S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)$. In particular, such a representation is unique.*
2. *$S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)$ if and only if S_i belongs to either $\mathcal{IS}(G(K_i), v)$ for some $v \in K_i \setminus K$, or $\overline{\mathcal{IS}}(G(K_i), K_i)$. In particular, S_i belongs to just one of them.*

Proof. The proof of the first claim is omitted here since it is similar to Lemma 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, $S_i \in \overline{\mathcal{IS}}(G(K_i), K \cap K_i)$.

Suppose that S_i is in $\overline{\mathcal{IS}}(G(K_i), K_i \cap K)$. When S_i includes a vertex v of $K_i \setminus K$, $S_i \in \mathcal{IS}(G(K_i), v)$. 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 $S_i \in \overline{\mathcal{IS}}(G(K_i), K_i)$. □

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

Equations 1. *Let G be a chordal graph and T be the clique tree of G. For a maximal clique K of G which is not a leaf of the clique tree, let $K_1, \ldots, K_l$ be the children of K in T. Furthermore, let $v \in K$. Then, the following identities hold.*

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

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

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

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

According to these equations, we obtain the following algorithm for counting the number of independent sets in a chordal graph G. For a maximal clique K of a chordal graph G, we denote the set of children of K in a clique tree of G by $\mathrm{CHD}(K)$.

Algorithm #IndSets

Input: A chordal graph $G = (V, E)$;

Output: The number of independent sets in G;

1. construct a clique tree of G by the algorithm in [16];
2. **call** #IndSetsIter(K_r);
3. **output** $\left|\overline{\mathcal{IS}}(G, K_r)\right| + \sum_{v \in K_r} |\mathcal{IS}(G(K_r), v)|$.

Procedure #IndSetsIter

Input: A maximal clique K of the chordal graph G;

4. If K is a leaf of the clique tree, set $|\overline{\mathcal{IS}}(G(K), K)| := 0$ and $|\mathcal{IS}(K, v)| := 1$ for each $v \in K$ and **return**;
5. **for each** child K' of K, **call** #IndSetsIter(K');
6. **for each** child K' of K, compute $\left|\overline{\mathcal{IS}}(G(K'), K' \cap K)\right|$ by $\left|\overline{\mathcal{IS}}(G(K'), K')\right| + \sum_{u \in K' \setminus K} |\mathcal{IS}(G(K'), u)|$;
7. compute $\left|\overline{\mathcal{IS}}(G(K), K)\right|$ by $\prod_{K' \in \mathrm{CHD}(K)} \left|\overline{\mathcal{IS}}(G(K'), K' \cap K)\right|$;
8. **for each** $v \in K$, compute $|\mathcal{IS}(G(K), v)|$ by
   $\prod_{K' \in \mathrm{CHD}(K), v \in K'} |\mathcal{IS}(G(K'), v)| \times \prod_{K' \in \mathrm{CHD}(K), v \notin K'} \left|\overline{\mathcal{IS}}(G(K'), K' \cap K)\right|$.

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

Proof. From Equations 1, the algorithm correctly computes the number of independent sets in a chordal graph G. 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 5 and 6 take $O(t(K'))$ and $O(|K_i|)$ time for each $K' \in \mathrm{CHD}(K)$ respectively. Step 7 can be done in $O(|\mathrm{CHD}(K)|)$. Next, we analyze the computation time for Step 8. Since $\left|\overline{\mathcal{IS}}(G(K), K)\right| = \prod_{K' \in \mathrm{CHD}(K)} \left|\overline{\mathcal{IS}}(G(K'), K' \cap K)\right|$, we have that $\prod_{K' \in \mathrm{CHD}(K), v \in K'} |\mathcal{IS}(G(K'), v)| \times \prod_{K' \in \mathrm{CHD}(K), v \notin K'} \left|\overline{\mathcal{IS}}(G(K'), K' \cap K)\right| = \left|\overline{\mathcal{IS}}(G(K), K)\right| \times \frac{\prod_{K' \in \mathrm{CHD}(K), v \in K'} |\mathcal{IS}(G(K'), v)|}{\prod_{K' \in \mathrm{CHD}(K), v \in K'} |\overline{\mathcal{IS}}(G(K'), K' \cap K)|}$, and we use this equation in Step 8. Then $|\mathcal{IS}(G(K), v)|$ can be computed in $O(|\{K' \mid K' \in \mathrm{CHD}(K), v \in K'\}|)$ time, thus Step 8 can be done in $O(\sum_{v \in K} |\{K' \mid K' \in \mathrm{CHD}(K), v \in K'\}|)$ 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 $O(\sum_{K \in T} (|K| + \sum_{v \in K} |\{K' \in \mathrm{CHD}(K) \mid v \in K'\}|))$, where T denotes the set of nodes in the clique tree. Since no two nodes give the same maximal clique of G, it holds that $O(\sum_{K \in T} |K|) = O(\sum_{v \in V} |C(v)|) = O(|V| + |E|)$ (remind that $C(v)$ denotes the clique induced by $N[v] \cap \{v' \mid v' \geq v$ in the perfect elimination ordering $\}$). Furthermore, it follows that $O(\sum_{K \in T} \sum_{v \in K} |\{K' \in \mathrm{CHD}(K) \mid v \in K'\}|) = O(\sum_{v \in V} |\{K' \in T \mid v \in K'\}|) = O(\sum_{v \in V} |C(v)|) = O(|V| + |E|)$. Hence, the total running time is $O(|V| + |E|)$. □

4 Efficient Algorithm for Counting 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 set of independent sets in G of size k. For a vertex v, let $\mathcal{IS}(G,v;k)$ be the set of independent sets in G of size k and including v, i.e., $\mathcal{IS}(G,v;k) = \{S \mid S \in \mathcal{IS}(G;k), v \in S\}$. For a vertex set U, let $\overline{\mathcal{IS}}(G,U;k)$ be the set of independent sets in G of size k and including no vertex of U, i.e., $\overline{\mathcal{IS}}(G,U;k) = \{S \mid S \in \mathcal{IS}(G;k), S \cap U = \emptyset\}$.

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.*

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

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

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

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

In contrast to Equations 1, the second and third equations of Equations 2 do not give straightforward ways to compute $|\mathcal{IS}(G(K),v;k)|$ and $\left|\overline{\mathcal{IS}}(G(K),K;k)\right|$, respectively, since we have to count how many combinations of $S_1,\ldots,S_l$ 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. For a vertex $v \in K$, we give indices to the children of K such that $K_1,\ldots,K_p$ include v and $K_{p+1},\ldots,K_l$ do not. For $k' \leq k$ and $l' \leq p$, let $Num(l';k') := \{S \mid S = \bigcup_{i=1}^{l'} S_i, S_i \in \mathcal{IS}(G(K_i),v), |S| = k'\}$. For $k' \leq k$ and $l' \geq p+1$, let $\overline{Num}(l';k') := \{S \mid S = \bigcup_{i=l'}^{l} S_i, S_i \in \overline{\mathcal{IS}}(G(K_i), K_i \cap K), |S| = k'\}$. Then, $|\mathcal{IS}(G(K),v;k)| = \sum_{h=0}^{k} |Num(p;h)| \times \left|\overline{Num}(p+1;k-h)\right|$.

For each l' and k', $|Num(l';k')|$ can be computed in $O(k')$ time based on the following recursive equation.

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

Similarly, $\left|\overline{Num}(l';k')\right|$ can be computed in $O(k')$ time. The computation of $|Num(l';k')|$ and $\left|\overline{Num}(l';k')\right|$ 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 $O(k^2|V|^2)$. In the following, we reduce the computation time by the technique used in the previous section.

Observe that $\left|\overline{\mathcal{IS}}(G(K),K;k')\right| = \sum_{h=0}^{k'} \left|\overline{Num}(p;h)\right| \times \left|\overline{Num}(p+1;k'-h)\right|$, which gives $\left|\overline{Num}(p+1;k')\right| \times \left|\overline{Num}(p;0)\right| = \left|\overline{\mathcal{IS}}(G(K),K;k')\right| - \sum_{h=1}^{k'} \left|\overline{Num}(p;h)\right| \times \left|\overline{Num}(p+1;k'-h)\right|$. This implies that we can compute $\left|\overline{Num}(k';p+1)\right|$ from $\left|\overline{\mathcal{IS}}(G(K),K;h)\right|$ and $\left|\overline{Num}(p;h)\right|$ 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 $k' = 0,\ldots,k$ in $O(k^2 \sum_{v \in K} |\{K' \mid K' \in \textsc{chd}(K), v \in K'\}|)$ 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 2. *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 number of independent sets of size k in a chordal graph $G = (V,E)$ can be computed in $O(|V|^2(|V| + |E|))$ time for all k from 0 to $|V|$.

5 Linear-Time Algorithm for Counting the Maximum Independent Sets

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

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

Equations 3. *With the same set-up as Equations 1, the following identities hold.*

$$\mathcal{MIS}(G(K)) = \mathrm{argmax}(\overline{\mathcal{MIS}}(G(K), K) \dot{\cup} \dot{\bigcup}_{v \in K} \mathcal{MIS}(G(K), v));$$

$$\mathcal{MIS}(G(K), v) = \mathrm{argmax}(\{S \mid S = \bigcup_{i=1}^{l} S_i, S_i \in \left\{ \begin{array}{ll} \mathcal{MIS}(G(K_i), v) & \textit{if } v \in K_i \\ \overline{\mathcal{MIS}}(G(K_i), K_i \cap K) & \textit{otherwise} \end{array} \right\}\});$$

$$\overline{\mathcal{MIS}}(G(K), K) = \mathrm{argmax}(\{S \mid S = \bigcup_{i=1}^{l} S_i, S_i \in \overline{\mathcal{MIS}}(G(K_i), K_i \cap K)\});$$

$$\overline{\mathcal{MIS}}(G(K_i), K_i \cap K) = \mathrm{argmax}(\overline{\mathcal{MIS}}(G(K_i), K_i) \dot{\cup} \dot{\bigcup}_{u \in K_i \setminus K} \mathcal{MIS}(G(K_i), u)).$$

Since the elements of the set 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. $N := 0$; $M := \max(\overline{\mathcal{MIS}}(G(K), K) \cup \bigcup_{v \in K} \mathcal{MIS}(G(K), v))$;
2. if the size of a member of $\overline{\mathcal{MIS}}(G(K), K)$ is equal to M, then $N := N + |\overline{\mathcal{MIS}}(G(K), K)|$;
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.

Therefore we have the following theorem.

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

6 Hardness of Counting the Maximal Independent Sets

So far, we have provided efficient algorithms for counting the number of independent sets, and so on, in a chordal graph. In this section, we turn to hardness results. First we consider the following counting problem: given a chordal graph G, count 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 4. *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 $\bigcup \mathcal{F} = X$. It is known that, given a finite set X and a set family $\mathcal{S}$, counting the number of set covers of X is #P-complete [14].

Proof of Theorem 4. The membership in #P of the problem is immediate. To show the #P-hardness, we reduce the problem to count the set covers to counting the 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 the counting problem of set covers. 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. It is easy to see that this construction can be done in polynomial time.

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 $2^{|\{S \in \mathcal{S} | x \notin S\}|}$.

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 $\sum_{x \in X} 2^{|\{S \in \mathcal{S} | x \notin S\}|}$. Since the last sum can be computed in polynomial time, this concludes the reduction. □

As a variation, let us consider the problem to count the number of minimum maximal independent sets in a given chordal graph. Note that a minimum maximal independent set in a chordal graph can be found in linear time [5]. In contrast to that, it is hard to count the number of minimum maximal independent sets in a chordal graph, as the following theorem tells.

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

Proof. We use the same reduction as in the proof of Theorem 4. 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. □

7 Hardness of Finding a Minimum Weight Maximal Independent Set

In this section, we consider an optimization problem to find a minimum weight maximal independent set in a chordal graph. 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 [5]. On the contrary, we show that the problem is actually hard when the weight is arbitrary.

Theorem 6. *Finding a minimum weight 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, which is known to be NP-hard.

Proof of Theorem 6. 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 4. 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 weight maximal independent set of G. Then, from the discussion in the proof of Theorem 4, 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 weight 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.

Theorem 7. *There is no randomized polynomial-time algorithm for finding a minimum weight maximal independent set in a chordal graph with approximation ratio $c \ln |V|$, for some fixed constant c, unless* NP $\subseteq$ ZTIME$(n^{O(\log \log n)})$.

Remark that ZTIME(t) is the class of languages which have a randomized algorithm running in expected time t with zero error.

To prove Theorem 7, we use the following restricted version of the minimum set cover problem: given a finite set X and a family $\mathcal{S} \subseteq 2^X$ such that $|A \cap B| \leq 1$ for every $A, B \in \mathcal{S}$, find a minimum set cover of X. We call this variant the *minimum set cover problem with intersection 1*.

Anil Kumar, Arya, & Ramesh showed that the minimum set cover problem with intersection 1 cannot be approximated by any randomized polynomial-time algorithm with approximation ratio $c' \ln |X|$, for some constant c', unless NP $\subseteq$ ZTIME$(n^{O(\log \log n)})$ [13]. We use this fact in our proof.

Before proving the theorem, we need a lemma which bounds the size of $\mathcal{S}$ in an instance of the minimum set cover problem with intersection 1. This is an easy special case of a theorem by Frankl & Wilson, which is well known in extremal combinatorics [6]. Hence we are not going to prove it.

Lemma 4. *Let X be a finite set and $\mathcal{S} \subseteq 2^X$ be a family of subsets of X such that $|A \cap B| \leq 1$ for every $A, B \in \mathcal{S}$. Then, $|\mathcal{S}| \leq |X| + 1$.*

Now we are ready to prove Theorem 7.

Proof of Theorem 7 (sketch). We basically follow the proof of Theorem 6, but this time, we use the minimum set cover problem with intersection 1. We also change the weight on the vertices. First set $\alpha := \lceil c \ln(3|X| + 2) \rceil |\mathcal{S}|$ and 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 $S' \in \mathcal{S}'$. Note that the number of vertices in our graph G is $|V| = |X| + 2|\mathcal{S}| \leq 3|X| + 2$ by Lemma 4. Then, we can argue that a maximal independent set in G with weight at least $c' \ln |V|$ times the weight of an optimum gives a set cover of size at least $c \ln |X|$ times the size of an optimum for a given instance of the minimum set cover problem with intersection 1. Then, this implies $\mathsf{NP} \subseteq \mathsf{ZTIME}(n^{O(\log \log n)})$ [13]. □

Remark that "$\mathsf{P} = \mathsf{NP}$" implies "$\mathsf{NP} \subseteq \mathsf{ZTIME}(n^{O(\log \log n)})$," but the converse is not known to be true. So, Theorems in this section are independent.

8 Enumeration

Due to space limitation, details of the following theorem are omitted here:

Theorem 8. *All independent sets in a chordal graph can be enumerated in constant time for each on average.* □

Similar algorithms can be developed for enumerating maximum independent sets, and independent sets of size k. The computation time of these algorithms can be reduced to constant time for each independent set.

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. M. Farber. Independent Domination in Chordal Graphs. *Operations Research Letters*, 1(4):134–138, 1982.
6. P. Frankl and R.M. Wilson. Intersection theorems with geometric consequences. *Combinatorica*, 1:357–368, 1981.
7. D.R. Fulkerson and O.A. Gross. Incidence Matrices and Interval Graphs. *Pacific J. Math.*, 15:835–855, 1965.
8. 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.
9. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.
10. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Annals of Discrete Mathematics 57. Elsevier, 2nd edition, 2004.
11. M.C. Golumbic and M. Lipshteyn. Chordal Probe Graphs. *Discrete Applied Mathematics*, to appear.
12. P.N. Klein. Efficient Parallel Algorithms for Chordal Graphs. *SIAM Journal on Computing*, 25(4):797–827, 1996.
13. V.S. Anil Kumar, Sunil Arya, and H. Ramesh. Hardness of Set Cover with Intersection 1. In *ICALP 2000*, pages 624–635. Lecture Notes in Computer Science Vol. 1853, Springer-Verlag, 2000.
14. 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.
15. 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.
16. J.P. Spinrad. *Efficient Graph Representations*. American Mathematical Society, 2003.
17. 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.