A Polynomial Delay Algorithm for Enumerating Minimal Dominating Sets in Chordal Graphs

Mamadou Moustapha Kanté[1], Vincent Limouzy[1], Arnaud Mary[2], Lhouari Nourine[1], and Takeaki Uno[3]

[1] Clermont-Université, Université Blaise Pascal, LIMOS, CNRS, France
[2] Université Claude Bernard Lyon 1 , LBBE, CNRS, France
[3] National Institute of Informatics, Japan

Abstract. An output-polynomial algorithm for the listing of minimal dominating sets in graphs is a challenging open problem and is known to be equivalent to the well-known Transversal problem which asks for an output-polynomial algorithm for listing the set of minimal transversals in hypergraphs. We give a polynomial delay algorithm to list the set of minimal dominating sets in chordal graphs, an important and well-studied graph class where such an algorithm was open[5]. The algorithm uses a new decomposition method of chordal graphs based on clique trees.

1 Introduction

A *hypergraph* $\mathcal{H}$ is a pair $(V, \mathcal{E})$ where V is a finite set and $\mathcal{E} \subseteq 2^V$ is called the set of *hyperedges*. Hypergraphs generalize graphs where each hyperedge has size at most 2. Given a hypergraph $\mathcal{H} := (V, \mathcal{E})$ and $\mathcal{C} \subseteq 2^V$, an *output-polynomial algorithm* for $\mathcal{C}$ is an enumeration algorithm for $\mathcal{C}$ whose running time is bounded by a polynomial depending on the sum of the sizes of $\mathcal{H}$ and $\mathcal{C}$. One of the central problem in the area of enumeration algorithm is the existence of an output-polynomial algorithm for the set of *minimal transversals* in hypergraphs, and is known as the *Transversal problem* or *Hypergraph dualization.* A *minimal transversal (or hitting set)* in a hypergraph $(V, \mathcal{E})$ is an inclusion-wise minimal subset T of V that intersects with every hyper-edge in $\mathcal{E}$. The transversal problem has several applications in artificial intelligence [8,9], game theory [14,20], databases [1,3,2], integer linear programming [3,2], to cite few. Despite the interest in Transversal problem the best known algorithm is the quasi-polynomial time algorithm by Fredman and Khachiyan[10] which runs in time $O(N^{\log(N)})$ where N is the cumulated size of the given hypergraph and its set of minimal transversals. However, there exist several classes of hypergraphs where an output-polynomial algorithm is known (see for instance [8,9,16] for some examples). Moreover, several particular subsets of vertices in graphs are special cases of transversals in hypergraphs and for some of them an output-polynomial algorithm is known, *e.g.* maximal independent sets, minimal vertex-covers, maximal (perfect) matchings, spanning trees, etc.

In this paper we are interested in the particular case of the Transversal problem, namely the enumeration of *minimal dominating sets* in graphs (Dom-Enum problem). A *minimal dominating set* in a graph is an inclusion-wise subset D of the vertex set such that every vertex is either in D or has a neighbor in D. In other words D is a minimal dominating set of

G if it is a minimal transversal of the *closed neighborhoods* of G, where the *closed neighborhood of a vertex x* is the set containing x and its neighbors. Since in important graph classes an output-polynomial algorithm for the DOM-ENUM problem is a direct consequence of already tractable cases for the Transversal problem, *e.g.* minor-closed classes of graphs, graphs of bounded degree, it is natural to ask whether an output-polynomial algorithm exists for the DOM-ENUM problem. However, it is proved in [16] that there exists an output-polynomial algorithm for the DOM-ENUM problem if and only if there exists one for the Transversal problem, and this remains true even if we restrict the DOM-ENUM problem to the co-bipartite graphs. This is surprising, but has the advantage of bringing tools from graph structural theory to this difficult problem and is particularly true for the DOM-ENUM problem since in several graph classes output-polynomial algorithms were obtained using the structure of the graphs: graphs of bounded clique-width [4], split graphs [16], interval and permutation graphs [17], line graphs [15,18], etc.

Since the DOM-ENUM problem in co-bipartite graphs is as difficult as the Transversal problem and co-bipartite graphs are a subclass of weakly chordal graphs, *i.e.* graphs with no cycles of length greater than or equal to 5, one can ask whether by restricting ourselves to graphs without cycles of length 4, which are exactly *chordal* graphs [7], one cannot expect an output-polynomial algorithm. In fact for several subclasses of chordal graphs an output-polynomial algorithm is already known, *e.g.* split graphs [16], undirected path graphs [15], chordal P_6-free. Furthermore, chordal graphs have a nice structure, namely the well-known *clique tree* which has been used to solve several algorithmic questions in chordal graphs. We prove the following.

Theorem 1. *There exists a polynomial delay algorithm for the* DOM-ENUM *problem in chordal graphs which uses polynomial space.*

An enumeration algorithm is *polynomial delay* if the maximum computation time between two outputs is polynomial in the input size, thus polynomial delay algorithm is output polynomial time. Notice that there exist problems where an output-polynomial algorithm is known and no polynomial delay algorithm exists unless P=NP [21].

Chordal graphs admit several linear structure (e.g. *perfect elimination ordering*) and tree structures called *clique trees.* The existence of these structures makes many problems polynomially solvable in chordal graphs. For example, using a clique tree we can split a chordal into several subgraphs by removing a clique. This decomposition leads to a dynamic programming algorithm for maximum independent set problem by considering the cases that each vertex of the clique is included in the independent set, since any independent set can include at most one vertex of the clique. However, dominating set may include several vertices in a clique, thus this approach is not applicable directly. To the best of our knowledge, there is no good way to deal with this difficulty, and this can explain why minimum dominating set problem is NP-complete. In this paper, we propose to use an "anti-chain" of cliques to decompose chordal graphs. The anti-chain decomposes a graph into several subgraphs, thus the solutions with respect to the anti-chain are obtained by the combination of the solutions of the subgraphs. Since the number of such subgraphs is limited, dynamic programming approach does work. This approach is more powerful than usual decomposition with cliques, in the sense that we can overcome the above difficulty when dealing with the minimal dominating set enumeration problem, thus, gives a new method for designing algorithms for chordal graphs.

2 Preliminaries

An algorithm is said to be *output-polynomial* if the running time is bounded by a polynomial in the input and output sizes. The delay is the maximum computation time between two outputs, pre-processing, and post-processing. If the delay is polynomial in the input size, the algorithm is called *polynomial delay.*

We refer to [6] for our graph terminology. We deal only with finite simple loopless undirected graphs. The vertex set of a graph G is denoted by V_G and its edge set by E_G. An edge between two vertices x and y is denoted by xy (yx respectively). Let G be a graph. The subgraph of G induced by $X \subseteq V_G$, denoted by $G[X]$ is the graph $(X, (X \times X) \cap E_G)$. For a vertex x of G we denote by $N_G(x)$ the set of neighbors of x, *i.e.* the set $\{y \in V_G \mid xy \in E_G\}$, and we let $N_G[x]$, the *closed neighborhood of* x, be $N_G(x) \cup \{x\}$. For $S \subseteq V_G$, let $N_G[S]$ denote $\bigcup_{x \in S} N_G[x]$. (We will remove the subscript when the graph is clear from the context and this will be the case for all sub or superscripts in the paper.) We say that a vertex x is *dominated* by a vertex y if $x \in N_G[y]$. A *dominating set* of G is a subset D of V_G such that every vertex of G is dominated by a vertex in D. A dominating set is *minimal* if it includes no other dominating set. For $D \subseteq V_G$, a vertex y is a *private neighbor* of $x \in D$ if $N_G[y] \cap D = \{x\}$; the set of private neighbors of a vertex $x \in D$ is denoted by $P(D, x)$. $D \subseteq V_G$ is an *irredundant set* of G if $P(D, x) \neq \emptyset$ for all $x \in D$. $D \subseteq V_G$ is a minimal dominating set of G if and only if D is a dominating set of G and D is an irredundant set.

A *clique* of G is a subset C of G that induces a complete graph, and a *maximal clique* is a clique C of G such that $C \cup \{x\}$ is not a clique for all $x \in V_G \backslash C$. We denote by $\mathcal{C}_G$ the set of maximal cliques of G.

For a *rooted tree* T, let us denote by $\preceq_T$ the relation where $u \preceq_T v$ if v is on the unique path from the root to u; if $u \preceq_T v$ then v is called an *ancestor* of u and u a *descendant* of v. Two nodes u and v of a rooted tree T are *incomparable* if $u \npreceq_T v$ and $v \npreceq_T u$. Given a node u of a rooted T the subtree of T rooted at u is the tree $T[\{v \in V_T \mid v \preceq_T u\}]$ which is rooted at u.

A graph G is called *chordal* if it does not contain chordless cycles of length greater than or equal to 4. From [12] with every chordal graph G, one can associate a tree that we denote by $\mathcal{T}_G$, called *clique tree*, whose nodes are the maximal cliques of G and such that for every vertex $x \in V_G$ the set $\mathcal{T}_G(x) := \{C \in V(\mathcal{T}_G) \mid \text{the maximal clique } C \text{ contains } x\}$ is a subtree of $\mathcal{T}_G$. Moreover, for every chordal graph G one can compute a clique tree in linear time (see for instance [11]). In the rest of the paper all clique trees are considered rooted.

Let $\mathcal{T}_G$ be a clique tree of a chordal graph G and let us denote its root by C_r. For each $C \in \mathcal{C}_G$, let us denote by $Pa(C)$ its parent and let $f(C) := C \backslash Pa(C)$, *i.e.*, the set of vertices in C that are not in any maximal clique C' ancestor of C. Notice that $\{f(C) \mid C \in \mathcal{T}_G\}$ is a partition of V_G. For each vertex $x \in V_G$, we denote by $C(x)$ the maximal clique C satisfying $x \in f(C)$. Notice that $C(x)$ is uniquely defined since exactly one maximal clique C satisfies $x \in f(C)$. For $C \in \mathcal{C}_G$, the subtree rooted at C is denoted by $\mathcal{T}_G(C)$, and the set of vertices $\bigcup_{C' \in \mathcal{T}_G(C)} f(C')$ is denoted by $V(C)$.

Property 1. Any clique tree $\mathcal{T}_G$ of a chordal graph G satisfies the following.
1. For each $C \in \mathcal{C}_G$, and each $x \in V_G \backslash V(C)$ either $(\{x\} \times f(C)) \subseteq E_G$ or $(\{x\} \times f(C)) \cap E_G = \emptyset$.
2. For any two incomparable C and C' in $\mathcal{C}_G$, we have $(f(C) \times f(C')) \cap E_G = \emptyset$.

For $S \subseteq V_G$ let $\mathcal{C}(S)$ denote the set $\{C(x) \mid x \in S\}$, $Up(S)$ the set of vertices x in V_G such that $C(x)$ is a proper ancestor of a clique $C \in \mathcal{C}(S)$ and $Uncov(S)$ be the vertex set $Up(S)\backslash N_G[S]$, *i.e.* the set of vertices in $Up(S)$ not dominated by S. For a vertex x, $Up(x)$ denotes $Up(\{x\})$. A subset $A \subseteq V_G$ is an *antichain* if (1) for any two vertices x and y in A we have $x \notin Up(y)$ and $y \notin Up(x)$, (2) for each vertex $z \in V_G\backslash Up(A)$, $A \cap (C(z) \cup Up(z)) \neq \emptyset$. Intuitively, A is an antichain if $\mathcal{C}(A)$ is a maximal set of pairwise incomparable maximal cliques. Given $S \subseteq V_G$, the *top-set* $A(S)$ is defined as the set of vertices of S included in the upmost cliques in $\mathcal{C}(S)$ that are not descendants of any other in $\mathcal{C}(S)$, i.e., $A(S) := \{x \in S \mid C(x) \text{ is in } \max_{\preceq \mathcal{T}}\{\mathcal{C}(S)\}\}$.

If $S \neq \emptyset$, let $\mathcal{L}(S)$ be the set of maximal cliques C satisfying (1) no descendant of C is in $\mathcal{C}(S)$, (2) some descendant of $Pa(C)$ is in $\mathcal{C}(S)$. In other words, $\mathcal{L}(S)$ is the set of upmost maximal cliques no descendant of which intersects with $\mathcal{C}(S)$, *i.e.*, $\mathcal{L}(S) := \max_{\preceq \mathcal{T}}\{C \in \mathcal{C}_G \mid C \text{ has no descendant in } \mathcal{C}(S)\}$. If $S = \emptyset$, let $\mathcal{L}(S)$ be $\{C_r\}$. We denote by $\mathcal{L}'(S)$ the set $\max_{\preceq \mathcal{T}}\{C' \in \mathcal{T}(C) \mid C \in \mathcal{L}(S) \text{ and } C' \cap S = \emptyset\}$.

We suppose that any clique tree $\mathcal{T}$ is numbered by a pre-order of the visit of a depth-first search. In this numbering, the numbers of the nodes in any subtree forms an interval of the numbers. It is worth noticing that this ordering is a linear extension of the descendant-ancestor relation. We say a clique is smaller than another clique when its number in the ordering is smaller than the other's. We also extend this numbering to the vertices of the corresponding graph so that the number of a vertex x is smaller than that of a vertex y if $C(x)$ is smaller than $C(y)$ (whenever $C(x) = C(y)$ we choose anyone). We also say that a vertex is smaller than another vertex if its number is smaller than the other's. For a vertex set S, $tail(S)$ denotes the largest vertex in S. A *prefix* of a vertex set S is its subset S' such that no vertex in $S\backslash S'$ is smaller than $tail(S')$. A *partial antichain* is a prefix of an antichain. We allow the $\emptyset$ to be a partial antichain.

Following this ordering of the vertices of a chordal graph G, a minimal dominating set D is said to be *greedily obtained* if we initially let $D := V_G$ and recursively apply the following rule: if D is not minimal, find the smallest vertex x in D such that $D\backslash\{x\}$ is a dominating set and set $D := D\backslash\{x\}$. Notice that given a graph G there is one greedily obtained minimal dominating set.

3 When Simplicity Means NP-Hardness

A typical way for the enumeration of combinatorial objects is the *backtracking* technique. We start from the emptyset, and in each iteration, we choose an element x, and partition the problem into two subproblems: the enumeration of those including x, and the enumeration of those not including x, and recursively solve these enumeration problems. If we can check the so called Extension Problem in polynomial time, then the algorithm is polynomial delay and uses only polynomial space. The Extension Problem is to answer the existence of an object including S and that does not intersect with X, where S is the set (partial solution) that we have already chosen in the ancestor iterations, and that includes all elements we decided to put in the output solution, and X is the set that we decided not to include in the output solution.

It is known that the Extension Problem for minimal dominating set enumeration is NP-complete [19], and one can even prove that it is still NP-complete in split graphs, which is a proper subclass of chordal graphs. However, split graphs have a good structure and in the paper [16], it is proved that if $S \cup X$ induces a clique the Extension Problem in split graphs can be solved in polynomial time and this combined with the structure of minimal dominating sets in split graphs lead to a polynomial delay algorithm for the Dom-Enum problem in split graphs. Chordal graphs also have a good tree structure induced by clique trees. Thus, by following this tree structure, the Extension Problem seems to be solvable. In precise, we consider the case in which a path $\mathcal{P}$, from the root, of the clique tree satisfies that both $V(C) \cap (S \cup X) \neq \emptyset$ and $V(C) \nsubseteq (S \cup X)$ holds only for cliques C included in $\mathcal{P}$. In other words, the condition is that for any clique $C \notin \mathcal{P}$ whose parent is in $\mathcal{P}$, either $V(C) \cap (S \cup X) = \emptyset$ (totally not determined) or $V(C) \subseteq (S \cup X)$ (totally determined) holds. The solutions are partially determined on the path $\mathcal{P}$, and thus the Extension Problem seems to be polynomial. However, Theorem 2 states that the problem is actually NP-complete.

Theorem 2. *The* Extension Problem *is* NP*-complete in chordal graphs even if a path $\mathcal{P}$, from the root, of the clique tree satisfies that any child C of a clique in $\mathcal{P}$ satisfies either $V(C) \cap (S \cup X) = \emptyset$ or $V(C) \subseteq (S \cup X)$.* □

To overcome these difficulties, we will follow another approach. In fact the NP-hardness comes from the fact that the root clique can have both un-dominated vertices and private neighbors of several vertices of S. In the following, we will introduce a new strategy for the enumeration, that repeatedly enumerates antichains in levelwise manner. Indeed for any minimal dominating set D of a chordal graph G, one can easily check that the set $A(D)$ is an antichain that moreover dominates $Up(A(D))$. Our strategy consists in enumerating such antichains and for each such antichain A enumerates the minimal dominating sets D such that $A(D) = A$. Let's be more precise in the forthcoming sections.

4 (K_1, K_2)-Extensions

From now on we consider a fixed chordal graph G and clique tree $\mathcal{T}$ of G with root C_r so that we do not need to recall them in the statements. Let $K_1, K_2 \subseteq C_r$ be given disjoint sets that are decided to be included in the solution. Intuitively, we are considering the subgraph induced by a subtree of the clique tree rooted at C_r, and K_1 and K_2 are vertices that we already decided to include in the solution, such that vertices in K_2 have private neighbors outside the subgraph, and vertices of K_1 do not. Without confusion we denote $K_1 \cup K_2$ by K. A (K_1, K_2)-*extension* of a partial antichain A is a vertex set D such that $(A \cup K) \subseteq D$ and $D \backslash (A \cup K) \subseteq \bigcup_{C \in \mathcal{L}(A \cup K)} V(C)$. Observe that if D is a (K_1, K_2)-extension of A, then A is a prefix of $A(D)$. When the partial antichain is not specified, (K_1, K_2)-extension is that for the empty partial antichain. A (K_1, K_2)-extension D is *feasible* if it is a dominating set and $P(D, x) \neq \emptyset$ for all $x \in D \backslash K_2$. A partial antichain A is (K_1, K_2)-*extendable* if it has a feasible (K_1, K_2)-extension.

Let us briefly explain the ideas of the algorithm and why we introduce (K_1, K_2)-extensions. We first observe that for any minimal dominating set D of G, its top-set is an $(\emptyset, \emptyset)$-extendable antichain. Moreover, $D \backslash A(D)$ is composed of vertices below $A(D)$, *i.e.*, any vertex in $D\backslash A(D)$ is included in $V(C)\backslash C$ for some $C \in \mathcal{C}(D)$. Using this, we can partition the minimal dominating sets according to their top-sets. Since these top-sets are $(\varnothing, \varnothing)$-extendable, we enumerate all $(\varnothing, \varnothing)$-extendable antichains, and for each $(\varnothing, \varnothing)$-extendable antichain A, enumerate all minimal dominating sets whose top-set is A. As by definition of (K_1, K_2)-extendable for some disjoint $K_1, K_2 \subseteq C_r$, for each $(\varnothing, \varnothing)$-antichain A there is at least one minimal dominating set whose top-set is A. Therefore, each output $(\varnothing, \varnothing)$-antichain will give rise to a solution. This is one of the key to polynomial delay.

Now for a minimal dominating set D and a clique $C \in \mathcal{C}(A(D))$, each vertex x in $D \cap (V(C) \cup C)$ cannot have a private neighbor in another $G[V(C') \cup C']$ for some other $C' \in \mathcal{C}(A(D))$. Therefore, we can treat each $G[V(C) \cup C]$ independently. However, for each $C \in \mathcal{C}(A(D))$ the set $D \cap (V(C) \cup C)$ is not necessarily a minimal dominating set of $G[V(C) \cup C]$ since $D \cap C$ may be equal to a singleton $\{x\}$ with x having a private neighbor in $Up(A(D))$. In such cases we are looking in $G[V(C) \cup C]$ a dominating set D' of $G[V(C) \cup C]$ containing x where x does not necessarily have a private neighbor, but all the other vertices in D' do, *i.e.*, D' is a feasible $(\varnothing, \{x\})$-extension in $G[V(C) \cup C]$ with clique tree $\mathcal{T}(C)$. This situation is what exactly motivated the notion of (K_1, K_2)-extensions.

Assume now we are given a pair (K_1, K_2) of disjoint sets in C_r and a (K_1, K_2)-extendable antichain A. Now contrary to $(\varnothing, \varnothing)$-antichains we can have a vertex x in $K := K_1 \cup K_2$ that belongs to several cliques in A. So we cannot independently make recursive calls in $G[V(C) \cup C]$ for each $C \in \mathcal{C}(A)$. But, for each feasible (K_1, K_2)-extension of A and each $C \in \mathcal{C}(A)$ the set $D \cap (V(C) \cup C)$ is a feasible (K_C^1, K_C^2)-extension of $G[V(C) \cup C]$ for some disjoint K_C^1 and K_C^2 in $(A \cup K) \cap C$. Now the whole task is to define for each $C \in \mathcal{C}(A)$ the sets K_C^1 and K_C^2 in $(A \cup K) \cap C$ in such a way that by combining all these feasible (K_C^1, K_C^2)-extensions we obtain a feasible (K_1, K_2)-extension of A, and also any feasible (K_1, K_2)-extension can be obtained in that way. Actually, the way of setting K_C^1 and K_C^2 is the key, and is described in the next section.

Let us prove some technical lemmas about (K_1, K_2)-extensions needed for proving the correctness of our algorithm. For $C \in \mathcal{C}_G$ and $x \in C$, let $\mathcal{F}(C, x) := \{C' \preceq_{\mathcal{T}} C \text{ and } C' \in \mathcal{L}'(x)\}$, and let $D_C(x)$ denote a vertex set composed of

1. $Z \subseteq V(C) \cap \left(\bigcup_{C' \in \mathcal{F}(C,x)} C' \right)$ such that $|Z \cap C'| = |Z \cap f(C')| = 1$ for all $C' \in \mathcal{F}(C, x)$,
2. a greedily obtained minimal dominating set of $G[(V(C)\backslash N_G[x])\backslash N_G[Z]]$.

If $x \notin C$, then we let $D_C(x)$ be a greedily obtained minimal dominating set of $G[V(C)]$.

Property 2. Let $C \in \mathcal{C}_G$ and let $x \in V_G$. Then $D_C(x)$ is an irredundant set in $G[V(C)]$ and every vertex in $V(C)\backslash N_G[x]$ is dominated by $D_C(x)$.

Given disjoint sets $K_1, K_2 \subseteq C_r$, $D \subseteq V_G \backslash K$ and $x \in D \cup K_1$, a vertex $y \in P(D \cup K, x)$ is said *safe* if either $x = y$, or the following two conditions are satisfied

(S1) $N_G(y) \cap V(C) \subseteq N_G[D_C(y)]$ for all $C \in \mathcal{L}'(D \cup K)$ with $y \in C$ and,
(S2) for each $z \in N_G[y] \cap Uncov(D \cup K)$, there is $C \in \mathcal{L}'(D \cup K)$ such that $z \in N_G[D_C(y)]$.

A vertex $x \in D$ is said *safe* if one of its private neighbors is safe.

Property 3. Let $x \in D \cup K_1$ and let $y \in P(D \cup K, x)$ be a safe for x. Then $V(C)\backslash\{y\} \subseteq N_G[D_C(y)]$ for all $C \in \mathcal{L}'(D \cup K)$ with $y \in C$.

Lemma 1. *Let A be a partial antichain and let $x \in A \cup K_1$. For $y \in P(A \cup K, x)$ that is non-safe, no (K_1, K_2)-extension D of A that is a dominating set satisfies that $y \in P(D, x)$.*

Proof. Since y is not safe, we have $x \neq y$, and therefore y violates one of the two conditions (S1) or (S2) to be safe. Suppose that (S1) is not satisfied, *i.e.* there is a clique $C \in \mathcal{L}'(A \cup K), y \in C$ such that there is a vertex z in $(N_G(y) \cap V(C))\backslash N_G[D_C(y)]$. Thus, any (K_1, K_2)-extension D of A that is a dominating set includes some vertices in $N_G[y]$ other than x, thus y is not a private neighbor of x.

Suppose now that (S2) is not satisfied, *i.e.* there is a vertex $z \in N_G[y] \cap Uncov(A \cup K)$ such that no clique $C \in \mathcal{L}'(A \cup K)$ satisfies $z \in N_G[D_C(y)]$. It implies from the definition of $D_C(y)$ that no vertex in $V(C)\backslash N_G[y]$ is adjacent to z in all cliques $C \in \mathcal{L}'(A \cup K)$. Thus, as in the previous case, in any (K_1, K_2)-extension D of A, y is not a private neighbor of x unless D is not a dominating set. □

Lemma 2. *Let A be a partial antichain and let $x \in A \cup K_1$ be safe. Then there is $y \in P(A \cup K, x)$ that is safe and such that $y \in V(C(x))$.*

Proof. The statement holds if $x \in P(A \cup K, x)$. If not, $C(x)$ includes another vertex in $A \cup K$, and it is adjacent to any vertex in $N_G[x]\backslash V(C(x))$ by Property 1. Thus all its safe private neighbors are always in $V(C(x))$. □

Lemma 3. *A partial antichain A is (K_1, K_2)-extendable if and only if the following two conditions are satisfied*

1. *any vertex in $Uncov(A \cup K)$ is included in a clique of $\mathcal{L}'(A \cup K)$,*
2. *all vertices in $A \cup K_1$ are safe.*

Proof. Let A be a (K_1, K_2)-extendable partial antichain. If (1) is not satisfied, there is a vertex $z \in Uncov(A \cup K)$ that is not included in any clique of $\mathcal{L}'(A \cup K)$, and by definition of (K_1, K_2)-extension no (K_1, K_2)-extension of A can dominate it. So (1) is always satisfied. Now, if (2) is not satisfied, there is a non-safe vertex x in $A \cup K_1$, thus all $y \in P(A \cup K, x)$ are non-safe. By Lemma 1 it follows that $P(D, x) = \varnothing$ for each (K_1, K_2)-extension D of A that is a dominating set, and then (2) is always satisfied.

Suppose now that the two conditions hold. For each $x \in A \cup K_1$ let us choose one safe private neighbor and let us denote the set of all these safe private neighbors by S. We consider a (K_1, K_2)-extension D generated from $A \cup K$ as follows. First of all notice that from the definition of private neighbor and safety for each $C \in \mathcal{L}'(A \cup K)$, $|C \cap S| \leqslant 1$. So, let $\mathcal{L}_1 := \{C \in \mathcal{L}'(A \cup K) \mid |C \cap S| = 1\}$ and $\mathcal{L}_0 := \{C \in \mathcal{L}'(A \cup K) \mid |C \cap S| = 0\}$. It is clear that $\{\mathcal{L}_0, \mathcal{L}_1\}$ is a bipartition of $\mathcal{L}'(A \cup K)$. Let $z \in S$. Now let

$$D := (A \cup K) \cup \left(\bigcup_{C \in \mathcal{L}_1, C \cap S = \{y\}} D_C(y) \right) \cup \left(\bigcup_{C \in \mathcal{L}_0} D_C(z) \right).$$

D is clearly a (K_1, K_2)-extension of A. By definition of $D_C(y)$ for each vertex $x \in D\backslash(A \cup K)$ we have that $P(D, x) \neq \varnothing$. It is moreover easy to check that for each $x \in A \cup K_1$, we have that $S \cap P(A \cup K, x) \in P(D, x)$. Thus, from Property 1, $P(D, x) \neq \varnothing$ for all $x \in D\backslash K_2$. Each vertex in $N_G[A \cup K]$ is dominated. Moreover, since for each $C \in \mathcal{L}_0$ we have $z \notin C$, by definition of $D_C(z)$ we have $V(C)$ is also dominated. Now, let $C \in \mathcal{L}_1$ and let $C \cap S = \{y\}$. We know from Property 2 that $V(C)\backslash N_G[y]$ is dominated by $D_C(y)$ and y is dominated by $A \cup K$ since y is safe for some vertex in $A \cup K_1$. So, it remains to show that $N_G(y) \cap V(C)$ is dominated. By the definition of safety we know that the two conditions (S1) and (S2) are satisfied, *i.e.* $N_G(y) \cap V(C)$ is dominated. □

As a corollary we have the following.

Lemma 4. *For any partial antichain A one can check in polynomial time whether A is (K_1, K_2)-extendable.*

Proof. By Lemma 3 it is enough to check if (1) all vertices in $A \cup K_1$ are safe and (2) each vertex in $Uncov(A \cup K)$ is included in a clique in $\mathcal{L}'(A \cup K)$. Since (2) can be easily checked in polynomial time from G and a clique tree of G, it remains to show that (1) can be checked in polynomial time. A vertex $x \in A \cup K_1$ is safe if either $x \in P(A \cup K_1, x)$ or there exists a safe $y \in V(C(x)) \cap P(A \cup K_1, x)$ by Lemma 2. But by the definition of safety for each $y \in V(C(x)) \cap P(A \cup K_1, x)$ the conditions (S1) and (S2) are of course checkable in polynomial time from G and a clique tree of G. □

5 The Algorithm

Our enumeration strategy is composed of nested enumerations: enumeration of (K_1, K_2)-extendable antichains, for each (K_1, K_2)-extendable antichain A and each $C \in \mathcal{C}(A)$ define K_C^1 and K_C^2 and enumerate all the feasible (K_C^1, K_C^2)-extensions, and finally the combinations of all these (K_C^1, K_C^2)-extensions. Since any minimal dominating set is a feasible extension of some $(\varnothing, \varnothing)$-extendable antichain, the completeness of the enumeration is trivial. The rest of the section is as follows. We first show how to enumerate (K_1, K_2)-extendable antichains for some fixed (K_1, K_2). Then we show, given a (K_1, K_2)-extendable antichain A, how to define K_C^1 and K_C^2 for each $C \in \mathcal{C}(A)$ and how to combine all the feasible (K_C^1, K_C^2)-extensions in order to obtain all feasible (K_1, K_2)-extensions of A. Before assuming that we can perform both tasks with polynomial delay and use only polynomial space let us show that we can enumerate with polynomial delay and polynomial space all the feasible (K_1, K_2)-extensions.

Enumeration of (K_1, K_2)-Extensions. The algorithm for enumerating all the feasible (K_1, K_2)-extensions, including the case of the root of the recursion, is composed of (K_1, K_2)-extendable antichain enumeration and of the enumeration of combinations of the feasible (K_C^1, K_C^2)-extensions for appropriate (K_C^1, K_C^2). It can be described as follows.

```
Algorithm EnumKExtension($G, \mathcal{T}, K_1, K_2$)
      $G$:graph, $\mathcal{T}$:clique tree
1. for each antichain $A$ output by EnumAntichain($G, \mathcal{T}, K_1, K_2, \varnothing$) do
2.    output each solution of EnumCombination($G, \mathcal{T}, K_1, K_2, A, A \cup K$)
3. end for
```

Assume that EnumAntichain($G, \mathcal{T}, K_1, K_2, \varnothing$) enumerates all (K_1, K_2)-extendable antichains (Lemma 5) and EnumCombination($G, \mathcal{T}, K_1, K_2, A, A \cup K$) enumerates all feasible (K_1, K_2)-extensions of A (Lemma 8), both with polynomial delay and use polynomial space. Then we have the following.

Theorem 3. *The call* EnumKExtension $(G, \mathcal{T}, K_1, K_2)$ *enumerates all feasible* (K_1, K_2)*-extensions in polynomial delay and uses polynomial space.*

Proof. By definition for every feasible (K_1, K_2)-extension D the top-set $A(D)$ is a (K_1, K_2)-extendable antichain. So by Lemmas 5 and 8 below every feasible (K_1, K_2)-extension is output. From the definition of (K_1, K_2)-extendable antichains every call in Step 1 outputs at least one feasible (K_1, K_2)-extension. Therefore, EnumKExtension $(G, \mathcal{T}, K_1, K_2)$ enumerates all feasible (K_1, K_2)-extensions. Now since EnumAntichain($G, \mathcal{T}, K_1, K_2, \varnothing$) and EnumCombination $(G, \mathcal{T}, K_1, K_2, A, A \cup K)$ run in polynomial delay with polynomial space we can conclude that EnumKExtension $(G, \mathcal{T}, K_1, K_2)$ runs in polynomial delay and use polynomial space. ▭

Enumeration of Antichains. Our strategy is to enumerate all (K_1, K_2)-extendable partial antichains by an ordinary backtracking algorithm, that repeatedly appends a vertex to the current solution that is larger than its tail. In this algorithm, any (K_1, K_2)-extendable partial antichain A is obtained from $A \backslash tail(A)$. Since $A \backslash tail(A)$ is a prefix of A, any (K_1, K_2)-extendable partial antichain is generated from another (K_1, K_2)-extendable partial antichain. This implies that the set of (K_1, K_2)-extendable partial antichains satisfies a kind of monotone property, and thus we can enumerate all (K_1, K_2)-extendable partial antichains with passing through only (K_1, K_2)-extendable partial antichains. The algorithm is described as follows.

```
Algorithm EnumAntichain($G, \mathcal{T}, K_1, K_2, A$)
      $G$:graph, $\mathcal{T}$:clique tree, $A$:$(K_1, K_2)$-extendable partial antichain
1. if $A$ is an antichain then output $A$;
2. for each vertex $z > tail(A)$ do
3.   if $A \cup \{z\}$ is a $(K_1, K_2)$-extendable partial antichain then
     call EnumAntichain($G, \mathcal{T}, K_1, K_2, A \cup \{z\}$)
4. end for
```

Lemma 5. *The call* EnumAntichain($G, \mathcal{T}, K_1, K_2, \varnothing$) *enumerates all* (K_1, K_2)*-extendable antichains in polynomial delay with polynomial space.*

Proof. We observe that for any (K_1, K_2)-extendable partial antichain A, $A \backslash tail(A)$ is a (K_1, K_2)-extendable partial antichain. Thus, one can easily prove by induction that the iteration inputting A is recursively called only by the iteration inputting $A \backslash tail(A)$. Therefore, all

(K_1, K_2)-extendable partial antichains are generated by this algorithm without repetition. For a (K_1, K_2)-extendable partial antichain A, there is at least one feasible (K_1, K_2)-extension D. By the definition of a feasible (K_1, K_2)-extension, $A(D\backslash K)$ is a (K_1, K_2)-extendable antichain with A as a prefix. This implies that at least one descendant of any iteration outputs an antichain, and every leaf of the recursion tree outputs an antichain. Then, the delay is bounded by the maximum computation time of an iteration multiplied by the depth of the recursion. The depth is at most $|V_G|$, thus the algorithm is polynomial delay since the loop at Step 2 runs at most n times and the (K_1, K_2)-extendability check can be done in polynomial time by Lemma 4. Since the depth is bounded by $|V_G|$, the algorithm uses obviously a polynomial space. □

Enumeration of Combinations. We now show, given a (K_1, K_2)-extendable antichain A, how to enumerate with polynomial delay and in polynomial space all feasible (K_1, K_2)-extensions of A by computing for each $C \in \mathcal{C}(A)$ all the (K^1_C, K^2_C)-extensions of $G[V(C) \cup C]$ for appropriate K^1_C and K^2_C and combining all of them. Note that the set A is the top-set of any feasible (K_1, K_2)-extension if and only if the (K_1, K_2)-extension is that of A. For pruning redundant partial combinations, we introduce the notion of a partial (K_1, K_2)-extension. A vertex set $D \supseteq A \cup K$ is called a *partial* (K_1, K_2)*-extension* of A if there is a feasible (K_1, K_2)-extension D' of A such that $D\backslash(A \cup K)$ is a prefix of $D'\backslash(A \cup K)$, and all the vertices in $V(C(x))$ for $x \in A$ is dominated by D if x is smaller than $tail(D\backslash(A \cup K))$. Our strategy is to enumerate all partial (K_1, K_2)-extensions of A, similar to the antichain enumeration. For a partial (K_1, K_2)-extension D of A, let $C^*(D)$ be the largest clique C in $\mathcal{C}(A)$ such that $(D\backslash(A \cup K)) \cap V(C) \neq \emptyset$, and $C_*(D)$ be the smallest clique C in $\mathcal{C}(A)$ such that a vertex in $V(C)$ is not dominated by D. Informally $C^*(D)$ is the last clique $C \in \mathcal{C}(A)$ such that $V(C)$ is dominated by D, and $C_*(D)$ the first clique in $\mathcal{C}(A)$ such that $V(C)$ is not dominated by D. To enumerate all partial (K_1, K_2)-extensions of A and find all (K_1, K_2)-extensions of A, we start from $D = A \cup K$ and repeatedly add a $(K^1_{C_*(D)}, K^2_{C_*(D)})$-extension of $G[V(C_*(D)) \cup C_*(D)]$ to D for appropriate $(K^1_{C_*(D)}, K^2_{C_*(D)})$, while keeping the extendability. To characterize the possible $(K^1_{C_*(D)}, K^2_{C_*(D)})$ we state the following lemma. Let $Q(C')$ be the vertices x in $K \cup A$ that has no safe private neighbor in $V(C) \cup C, C > C'$, and none of its private neighbor in $P(K \cup A \cup D, x)$ is included in $Up(A)\backslash C'$ or in $V(C), C < C'$. In other words $Q(C')$ is the set of vertices in $K \cup A$ that we must give a private neighbor in $V(C') \cup C'$ for any (K_1, K_2)-extension of A containing D.

Lemma 6. *For a non-empty partial (K_1, K_2)-extension D, $D \cap (V(C^*(D)) \cup C^*(D))$ is a feasible (K'_1, K'_2)-extension in $G[V(C^*(D)) \cup C^*(D)]$ where $K'_1 = Q(C^*(D))$ and $K'_2 = ((A \cup K) \cap C^*(D))\backslash K'_1$.*

Proof. By definitions of partial (K_1, K_2)-extension and of C^*, $D \cap (V(C^*(D) \cup C^*(D))$ dominates $V(C^*(D))$. Moreover, every vertex x in $Q(C^*(D))$ has a private neighbor only in $V(C^*(D)) \cup C^*(D)$, and moreover $x \in C^*(D)$. Thus, the statement holds. □

Lemma 7. *Let D be a partial (K_1, K_2)-extension of A and suppose that $C_*(D)$ exists. For any feasible (K'_1, K'_2)-extension D' in $G[V(C_*(D)) \cup C_*(D)]$ where $K'_1 = Q(C_*(D))$, $K'_2 = ((A \cup K) \cap C_*(D))\backslash K'_1$, $D \cup D'$ is a partial (K_1, K_2)-extension of A.*

Proof. As in the proof of Lemma 3, we choose one private neighbor for vertices in $A \cup K$ that have safe private neighbors in $V(C), C > C_*(D)$ and let S be the set of these selected vertices. Then we let $\mathcal{L}_1 := \{C \in \mathcal{L}'(A \cup K) \mid C > C_*(D), |C \cap S| = 1\}$ and $\mathcal{L}_0 := \{C \in \mathcal{L}'(A \cup K) \mid C > C_*(D), |C \cap S| = 0\}$. Let $z \in S$. Now let

$$D^* := (A \cup K \cup D \cup D') \cup \left(\bigcup_{C \in \mathcal{L}_1, C \cap S = \{y\}} D_C(y) \right) \cup \left(\bigcup_{C \in \mathcal{L}_0} D_C(z) \right).$$

According to the proof of Lemma 3, D^* is a feasible (K_1, K_2)-extension of A. □

We can now describe the algorithm.

```
Algorithm EnumCombination($G, \mathcal{T}, K_1, K_2, A, D$)
      $G$:graph, $\mathcal{T}$:clique tree, $A$:$(K_1, K_2)$-extendable antichain
      $D$: a partial $(K_1, K_2)$-extension of $A$
1. if  $C_*(D)$ does not exist then output $D$; return
2. $K'_1 = Q(C_*(D))$, $K'_2 := ((A \cup K) \cap C_*(D)) \backslash K'_1$
3. for each $D'$ output by EnumKExtension($G[V(C_*(D)) \cup C_*(D)], \mathcal{T}(C^*(D)), K'_1, K'_2$)
4.    call EnumCombination($G, \mathcal{T}, K_1, K_2, A, D \cup D'$)
5. end for
```

Lemma 8. *The call* EnumCombination$(G, \mathcal{T}, K_1, K_2, A, A \cup K)$ *enumerates all feasible* (K_1, K_2)*-extensions whose top-set is* A *in polynomial delay and uses polynomial space.*

Proof. From Lemma 6, the iteration of a partial (K_1, K_2)-extension D of A is generated only from the iteration of $D \backslash (V(C^*(D) \backslash C^*(D)))$. This assures that the algorithm enumerates all partial (K_1, K_2)-extensions of A without duplication. From Lemma 7, there is at least one feasible (K_1, K_2)-extension D' of A including the partial (K_1, K_2)-extension D of A that is the input of the iteration. Thus, all the leaf iterations of the recursion of this algorithm always outputs a feasible (K_1, K_2)-extension of A. Now the delay is bounded by the maximum computation time of an iteration multiplied by the depth of the recursion. The depth is at most $|V_G|$, thus the algorithm is polynomial delay since EnumKExtension runs with polynomial delay. Since the depth is at most $|V_G|$, the algorithm is obviously polynomial space. □

Proof (of Theorem 1). By definition every minimal dominating set of G is a feasible $(\emptyset, \emptyset)$-extension. Therefore, the call EnumKExtension $(G, \mathcal{T}, \emptyset, \emptyset)$ enumerates all minimal dominating sets in polynomial delay and polynomial space by Theorem 3. □

6 Conclusion

We have proved that one can list all the minimal dominating sets of a chordal graph with polynomial delay and polynomial space. The result enlarged the classes in that minimal dominating set enumeration is output-polynomially solvable. However, the problem is still open for several graph classes such as bipartite graphs and unit-disk graphs. In particular, chordal bipartite graph admits an output-polynomial algorithm[13]. Applying our decomposition technique to chordal bipartite is an interesting future research.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.
2. E. Boros, V. Gurvich, L. Khachiyan, and K. Makino. Dual-bounded generating problems: Partial and multiple transversals of a hypergraph. *SIAM J. Comput.*, 30(6):2036–2050, 2000.
3. E. Boros, V. Gurvich, L. Khachiyan, and K. Makino. Generating weighted transversals of a hypergraph. In *RUTGERS UNIVERSITY*, pages 13–22, 2000.
4. B. Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 157(12):2675–2700, 2009.
5. Jean-François Couturier, Pinar Heggernes, Pim van't Hof, and Dieter Kratsch. Minimal dominating sets in graph classes: Combinatorial bounds and enumeration. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and Gy"orgy Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, pages 202–213. Springer Berlin Heidelberg, 2012.
6. R. Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, 2005.
7. G.A. Dirac. On rigid circuit graphs. *Abhandlungen Aus Dem Mathematischen Seminare der Universität Hamburg*, 25(1-2):71–76, 1961.
8. T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM J. Comput.*, 24(6):1278–1304, 1995.
9. T. Eiter, G. Gottlob, and K. Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM J. Comput.*, 32(2):514–537, 2003.
10. Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618 – 628, 1996.
11. P. Galinier, M. Habib, and C. Paul. Chordal graphs and their clique graphs. In Manfred Nagl, editor, *WG*, volume 1017 of *Lecture Notes in Computer Science*, pages 358–371. Springer, 1995.
12. F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47 – 56, 1974.
13. P. A. Golovach, P. Heggernes, M. M. Kanté, D. Kratsch, and Y. Villanger. Enumerating minimal dominating sets in chordal bipartite graphs. *Discrete Applied Mathematics*, 2015. Available online at http://dx.doi.org/10.1016/j.dam.2014.12.010.
14. V.A. Gurvich. On theory of multistep games. *{USSR} Computational Mathematics and Mathematical Physics*, 13(6):143 – 161, 1973.
15. M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. On the neighbourhood helly of some graph classes and applications to the enumeration of minimal dominating sets. In *ISAAC*, volume 7676 of *LNCS*, pages 289–298. Springer, 2012.
16. M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM J. Discrete Math.*, 28(4):1916–1929, 2014.
17. M. M. Kanté, V. Limouzy, A. Mary, L. Nourine, and T. Uno. On the enumeration and counting of minimal dominating sets in interval and permutation graphs. In *ISAAC*, volume 8283 of *LNCS*, pages 339–349. Springer, 2013.
18. M. M. Kanté, V. Limouzy, A. Mary, L. Nourine, and T. Uno. Polynomial delay algorithm for listing minimal edge dominating sets in graphs. *CoRR*, abs/1404.3501, 2014. To appear in WADS'15.
19. A. Mary. *Énumération des Dominants Minimaux d'un graphe.* PhD thesis, Université Blaise Pascal, 2013.
20. K.G. Ramamurthy. *Coherent Structures and Simple Games.* Theory and decision library, Game theory, mathematical programming, and operations research: Series C. Springer, 1990.

21. Y. Strozecki. *Enumeration Complexity and Matroid Decomposition*. PhD thesis, Université Paris Diderot - Paris 7, 2010.

Appendix

Proposition 1. *The* Extension Problem *is* NP*-complete in split graphs.*

Proof. It is proved in [19] that the following problem is NP-complete: Given G and $A \subset V_G$ decide whether there exists a minimal dominating set of G containing A. We reduce it to the Extension Problem in split graphs. Let G be a graph, and let $V'_G := \{x' \mid x \in V_G\}$ a disjoint copy of V_G. We let $Split(G)$ be the split graph with vertex set $V_G \cup V'_G$ where V_G and V'_G are respectively the clique and the independent set in $Split(G)$; now xy' is an edge if $x \in N_G[y]$. Now it is easy to check that asking whether there exists a minimal dominating set of G that contains $A \subset V_G$ is equivalent to asking whether there exists a minimal dominating set of $Split(G)$ that contains A and does not intersect with $V'_G \backslash A'$ where $A' := N_{Split(G)}[A] \cap V'_G$. □

Proof (of Theorem 2). We reduce Sat to our problem. Let φ be an instance of Sat with $x_1, \ldots, x_n$ the variables and $c_1, \ldots, c_m$ the clauses of φ. We construct a chordal graph as follows. The vertex set of the graph is

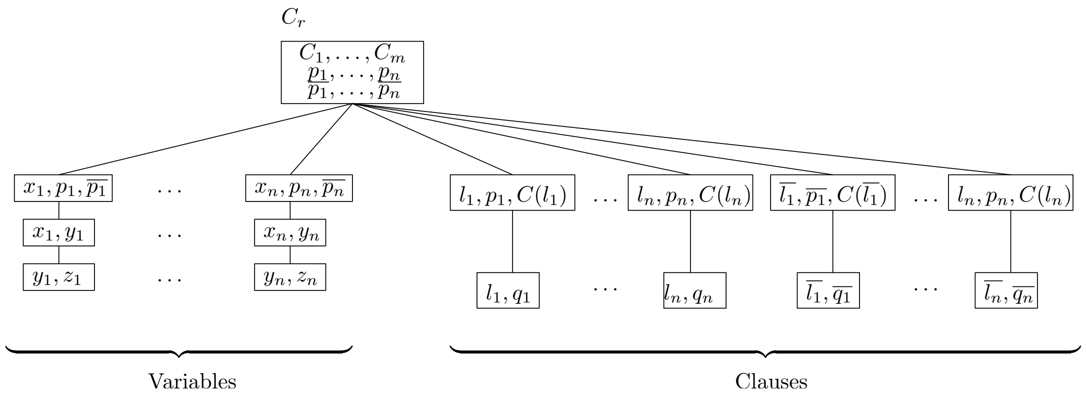
$$\{x_1, \ldots, x_n, c_1, \ldots, c_m, p_1, \ldots, p_n, \bar{p}_1, \ldots, \bar{p}_n, l_1, \ldots, l_n\} \bigcup$$
$$\{\bar{l}_1, \ldots, \bar{l}_n, y_1, \ldots, y_n, z_1, \ldots, z_n, q_1, \ldots, q_n, \bar{q}_1, \ldots, \bar{q}_n\},$$

where l_i and $\bar{l}_i$ are literals representing respectively x_i and $\bar{x_i}$ (notice that if one literal does not appear, the corresponding vertex is not created). Since with every clique tree one can associate a unique chordal graph, we will construct the clique tree of the chordal graph. For each $1 \leqslant i \leqslant n$, we let $C(l_i)$ and $C(\bar{l}_i)$ be the set of clauses containing the literal l_i and $\bar{l}_i$ respectively. We let its root be $C_r := \{c_1, \ldots, c_m, p_1, \ldots, p_n, \bar{p}_1, \ldots, \bar{p}_n\}$. The other maximal cliques are defined as follows. For each $1 \leqslant i \leqslant n$, we let $C_{x_i} = \{x_i, p_i, \bar{p}_i\}$, $C_{y_i} = \{y_i, x_i\}$, $C_{z_i} = \{y_i, z_i\}$, $C_{q_i} = \{q_i, l_i\}$, $C_{\bar{q}_i} = \{\bar{q}_i, \bar{l}_i\}$, $C_{l_i} = \{l_i, p_i\} \cup C(l_i)$, and $C_{\bar{l}_i} = \{\bar{l}_i, \bar{p}_i\} \cup C(\bar{l}_i)$ with the following parent-child relation: C_{x_i}, C_{l_i} and $C_{\bar{l}_i}$ are the children of C_r, C_{y_i} is the only child of C_{x_i} and C_{z_i} is the only child of C_{y_i}, C_{q_i} and $C_{\bar{q}_i}$ are the only children of C_{l_i} and $C_{\bar{l}_i}$ respectively. It is easy to check that the constructed tree is indeed a clique tree. See Figure 1 for an illustration.

We set $S := \{x_1, \ldots, x_n, y_1, \ldots, y_n\}$ and $X := \{z_1, \ldots, z_n, p_1, \ldots, p_n, \bar{p}_1, \ldots, \bar{p}_n\} \cup \{c_1, \ldots, c_m\}$ and $\mathcal{P} := \{C_r\}$. For each $1 \leqslant i \leqslant n$, we have by construction $V(C_{x_i}) \subseteq S \cup X$, and $(V(C_{l_i}) \cup V(C_{\bar{l}_i})) \cap (S \cup X) = \varnothing$. Therefore, for any maximal clique C child of C_r, either $V(C) \cap (S \cup X) = \varnothing$, or $V(C) \subseteq (S \cup X)$ holds, thus the condition of the statement holds.

One can easily check that any satisfiable assignment of φ leads to a minimal dominating set containing S and that does not intersect X. Let us prove the converse direction. We observe when we choose both l_i and $\bar{l}_i$ in the dominating set, x_i loses its private neighbors. Thus, any minimal dominating set can include at most one of them. On the other hand, exactly one of l_i and q_i (resp., $\bar{l}_i$ and $\bar{q}_i$) must be included in any minimal dominating set, so that it dominates l_i and q_i (resp., $\bar{l}_i$ and $\bar{q}_i$), and both must be private neighbors of the chosen one. Moreover, to dominate each clause c_j, at least one literal of c_j has to be included in any minimal dominating set. Hence, for any minimal dominating set D including S and not intersect with X, the set of literals included in D corresponds to a satisfiable assignment.

Therefore, the answer of the Extension Problem is yes if and only if φ has a satisfiable assignment. □



C_r
$C_1, \dots, C_m$
$\underline{p_1}, \dots, \underline{p_n}$
$p_1, \dots, p_n$
$x_1, p_1, \overline{p_1}$
$x_n, p_n, \overline{p_n}$
x_1, y_1
x_n, y_n
y_1, z_1
y_n, z_n
$l_1, p_1, C(l_1)$
$l_n, p_n, C(l_n)$
$\overline{l_1}, \overline{p_1}, C(\overline{l_1})$
$l_n, p_n, C(l_n)$
l_1, q_1
l_n, q_n
$\overline{l_1}, \overline{q_1}$
$\overline{l_n}, \overline{q_n}$
Variables
Clauses

Fig. 1. An illustration of the construction of Theorem 2.

Proof (of Property 2). We first prove that $D_C(x)$ is irredundant. Since each minimal dominating set is also an irredundant set, we can assume that $x \in C$. By definition of Z we have that $\{x\} \times Z \cap E_G = \varnothing$. Moreover, by Property 1(2) no two vertices of Z are adjacent. Since by construction of $D_C(x)\backslash Z$ no vertex in $D_C(x)\backslash Z$ is adjacent to a vertex of Z, we can conclude that for each $z \in Z$ we have $z \in P(D_C(x), z)$. Moreover, since $(D_C(x)\backslash Z) \cap N_G[Z] = \varnothing$ and $D_C(x)\backslash Z$ is a minimal dominating set of $G[(V(C)\backslash N_G[x])\backslash N_G[Z]]$, we can conclude that $P(D_C(x), y) \neq \varnothing$ for all $y \in (D_C(x)\backslash N_G[x])\backslash N_G[Z]$.

Let us now prove that $V(C)\backslash N_G[x]$ is dominated by $D_C(x)$. If $x \notin C$, then $D_C(x)$ is a minimal dominating set of $G[V(C)]$ and then we are done. So, assume that $x \in C$ and let $y \in V(C)\backslash N_G[x]$. Then $C(y)$ is necessarily a descendant of a clique $C' \in \mathcal{L}'(x)$ and such that $C' \preceq_{\mathcal{T}} C$. So, either $y \in N_G[Z]$ or $y \notin N_G[Z]$. In both cases, it is dominated by $D_C(x)$. □

Proof (of Property 3). By Property 2 $V(C)\backslash N_G[y]$ is dominated by $D_C(y)$. By definition of safety $N_G(y)$ is dominated by $D_C(y)$. Therefore $V(C)\backslash\{y\}$ is dominated by $D_C(y)$ for all $C \in \mathcal{L}'(D \cup K)$ with $y \in C$. □