Efficient Enumeration of Dominating Sets for Sparse Graphs

Kazuhiro Kurita
IST, Hokkaido University, Sapporo, Japan
k-kurita@ist.hokudai.ac.jp

Kunihiro Wasa
National Institute of Informatics, Tokyo, Japan
wasa@nii.ac.jp
https://orcid.org/0000-0001-9822-6283

Hiroki Arimura
IST, Hokkaido University, Sapporo, Japan
arim@ist.hokudai.ac.jp

Takeaki Uno
National Institute of Informatics, Tokyo, Japan
uno@nii.ac.jp

Abstract

A dominating set D of a graph G is a set of vertices such that any vertex in G is in D or its neighbor is in D. Enumeration of minimal dominating sets in a graph is one of central problems in enumeration study since enumeration of minimal dominating sets corresponds to enumeration of minimal hypergraph transversal. However, enumeration of dominating sets including non-minimal ones has not been received much attention. In this paper, we address enumeration problems for dominating sets from sparse graphs which are degenerate graphs and graphs with large girth, and we propose two algorithms for solving the problems. The first algorithm enumerates all the dominating sets for a k-degenerate graph in $O(k)$ time per solution using $O(n+m)$ space, where n and m are respectively the number of vertices and edges in an input graph. That is, the algorithm is optimal for graphs with constant degeneracy such as trees, planar graphs, H-minor free graphs with some fixed H. The second algorithm enumerates all the dominating sets in constant time per solution for input graphs with girth at least nine.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Enumeration algorithm, polynomial amortized time, dominating set, girth, degeneracy

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.8

1 Introduction

One of the fundamental tasks in computer science is to enumerate all subgraphs satisfying a given constraint such as cliques [23], spanning trees [25], cycles [2], and so on. One of the approaches to solve enumeration problems is to design exact exponential algorithms, i.e., *input-sensitive* algorithms. Another mainstream of solving enumeration problems is to design *output-sensitive* algorithms, i.e., the computation time depends on the sizes of both of an input and an output. An algorithm $\mathcal{A}$ is *output-polynomial* if the total computation time is polynomial of the sizes of input and output. $\mathcal{A}$ is an *incremental polynomial time algorithm* if the algorithm needs $O(poly(n,i))$ time when the algorithm outputs the i^{th} solution after outputting the $(i-1)^{\text{th}}$ solution, where $poly(\cdot)$ is a polynomial function. $\mathcal{A}$

runs in *polynomial amortized time* if the the total computation time is $O(poly(n)N)$, where n and N are respectively the sizes of an input and an output. In addition, $\mathcal{A}$ runs in *polynomial delay* if the maximum interval between two consecutive solutions is $O(poly(n))$ time and the preprocessing and postprocessing time is $O(poly(n))$. From the point of view of tractability, efficient algorithms for enumeration problems have been widely studied [1, 2, 6, 11, 12, 20, 23, 25, 27]. On the other hands, Lawler *et al.* show that some enumeration problems have no output-polynomial time algorithm unless $P = NP$ [21]. In addition, recently, Creignou *et al.* show a tool for showing the hardness of enumeration problems [8].

A dominating set is one of a fundamental substructure of graphs and finding the minimum dominating set problem is a classical NP-hard problem [12]. A vertex set D of a graph G is a dominating set of G if every vertex in G is in D or has at least one neighbors in D. The enumeration of *minimal* dominating sets of a graph is closely related to the enumeration of *minimal hypergraph transversals* of a hypergraph [10]. Kanté *et al.* [18] show that the minimal dominating set enumeration problem and the minimal hypergraph transversal enumeration problem are equivalent, that is, the one side can be solved in output-polynomial time if the other side can be also solved in output-polynomial time. Several algorithms that run in polynomial delay have been developed when we restrict input graphs, such as permutation graphs [18], chordal graphs [19], line graphs [20], graphs with bounded degeneracy [16], graphs with bounded tree-width [7], graphs with bounded clique-width [7], and graphs with bounded (local) LMIM-width [14]. Incremental polynomial-time algorithms have also been developed, such as chordal bipartite graphs [13], graphs with bounded conformality [3], and graphs with girth at least seven [15]. Kanté *et al.* [17] show that the conformality of the closed neighbourhood hypergraphs of line graphs, path graphs, and (C_4, C_5, claw)-free graphs is constant. However, it is still open whether there exists an output-polynomial time algorithm for enumerating minimal dominating sets from general graphs.

Since the number of solutions exponentially increases compared to the minimal version, even if we can develop an enumeration algorithm that runs in constant time per solution, the algorithm becomes theoretically much slower than some enumeration algorithm for minimal dominating sets. However, when we consider the real-world problem, we sometimes use another criteria for enumerating solutions that form dominating sets in a graph. That is, enumeration algorithms for minimal dominating sets may not fit in with other variations of minimal domination problems. E.g., a tropical dominating set [9] and a rainbow dominating set [4] are such a dominating set. Thus, when we enumerate solutions of such domination problems, our algorithm becomes a base-line algorithm for these problems. Thus, our main goal is to develop an efficient enumeration algorithm for dominating sets.

Main results: In this paper, we consider the relaxed problems, i.e., enumeration of all dominating sets that include non-minimal ones in a graph. We present two algorithms, `EDS-D` and `EDS-G`. `EDS-D` enumerates all dominating sets in $O(k)$ time per solution, where k is the degeneracy of a graph (Theorem 13). Moreover, `EDS-G` enumerates all dominating sets in constant time per solution for a graph with girth at least nine (Theorem 25), where the girth is the length of minimum cycle in the graph.

By straightforwardly using an enumeration framework such as the reverse search technique [1], we can obtain an enumeration algorithm for the problem that runs in $O(n)$ or $O(\Delta)$ time per solution, where n and Δ are respectively the number of vertices and the maximum degree of an input graph. Although dominating sets are fundamental in computer science, no enumeration algorithm for dominating sets that runs in strictly faster than such a trivial algorithm has been developed so far. Thus, to develop efficient algorithms, we focus

on the *sparsity* of graphs as being a good structural property and, in particular, on the *degeneracy* and *girth*, which are the measures of sparseness. As our contributions, we develop two optimal algorithms for enumeration of dominating sets in a sparse graph. We first focus on the degeneracy of an input graph. A graph is k-degenerate [22] if any subgraph of the graph has a vertex whose degree is at most k. The degeneracy of a graph is the minimum value of k such that the graph is k-degenerate. Note that $k \leq \Delta$ always holds. It is known that some graph classes have constant degeneracy, such as forests, grid graphs, outerplanar graphs, planer graphs, bounded tree width graphs, and H-minor free graphs for some fixed H [5,26]. A k-degenerate graph has a *good* vertex ordering, called a *degeneracy ordering* [24], as shown in Section 3. So far, this ordering has been used to develop efficient enumeration algorithms [6, 11, 27]. By using this ordering and the reverse search technique [1], we show that our proposed algorithm `EDS-D` can solve the relaxed problem in $O(k)$ time per solution. This implies that `EDS-D` can optimally enumerate all the dominating sets in an input graph with constant degeneracy.

We next focus on the girth of a graph. Enumeration of minimal dominating sets can be solved efficiently if an input graph has no short cycles since its connected subgraphs with small diameter form a tree. Indeed, this local tree structure has been used in minimal dominating sets enumeration [15]. For the relaxed problem, by using the reverse search technique, we can easily show that the delay of our proposed algorithm `EDS-G` for general graphs is $O(\Delta^3)$ time. However, if an input graph has the large girth, then each recursive call generates enough solutions, that is, we can amortize the complexity of `EDS-G`. Thus, by amortizing the time complexity using this local tree structure, we show that the problem can be solve in constant time per solution for graphs with girth at least nine.

2 A Basic Algorithm Based on Reverse Search

Let $G = (V(G), E(G))$ be a simple undirected graph, that is, G has no self loops and multiple edges, with vertex set $V(G)$ and edge set $E(G)$ is a set of pairs of vertices. If no confusion arises, we will write $V = V(G)$ and $E = E(G)$. Let u and v be vertices in G. An edge e with u and v is denoted by $e = \{u, v\}$. u and v are *adjacent* if $\{u, v\} \in E$. We denote by $N_G(u)$ the set of vertices that are adjacent to u on G and by $N_G[u] = N_G(u) \cup \{u\}$. We say v is a *neighbor* of u if $v \in N_G(u)$. The *set of neighbors* of U is defined as $N(U) = \bigcup_{u \in U} N_G(u) \setminus U$. Similarly, let $N[U]$ be $\bigcup_{u \in U} N_G(u) \cup U$. Let $d_G(v) = |N_G(v)|$ be the *degree* of u in G. We call the vertex v *pendant* if $d_G(v) = 1$. $\Delta(G) = \max_{v \in V} d(v)$ denotes the maximum degree of G. A set X of vertices is a *dominating set* if X satisfies $N[X] = V$.

For any vertex subset $V' \subseteq V$, we call $G[V'] = (V', E[V'])$ an *induced subgraph* of G, where $E[V'] = \{\{u, v\} \in E(G) \mid u, v \in V'\}$. Since $G[V']$ is uniquely determined by V', we identify $G[V']$ with V'. We denote by $G \setminus \{e\} = (V, E \setminus \{e\})$ and $G \setminus \{v\} = G[V \setminus \{v\}]$. For simplicity, we will use $v \in G$ and $e \in G$ to refer to $v \in V(G)$ and $e \in E(G)$, respectively.

We now define the dominating set enumeration problem as follows:

▶ **Problem 1.** *Given a graph G, then output all dominating sets in G without duplication.*

In this paper, we propose two algorithms `EDS-D` and `EDS-G` for solving Problem 1. These algorithms use the degeneracy ordering and the local tree structure, respectively. Before we enter into details of them, we first show the basic idea for them, called *reverse search method* that is proposed by Avis and Fukuda [1] and is one of the framework for constructing enumeration algorithms.

An algorithm based on reverse search method enumerates solutions by traversing on an implicit tree structure on the set of solution, called a *family tree*. For building the family tree,

Algorithm 1: `EDS` enumerates all dominating sets in amortized polynomial time.

```
1 Procedure EDS(G = (V, E))                                  // G: an input graph
2 |  AllChildren(V, V, G);
3 Procedure AllChildren(X, C(X), G = (V, E))  // X: the current solution
4 |  Output X;
5 |  for v ∈ C(X) do
6 |  |  Y ← X \ {v}; C(Y) ← {u ∈ C(X) | N[Y \ {u}] = V ∧ P(Y \ {u}) = Y};
7 |  |  AllChildren(Y, C(Y), G);
```

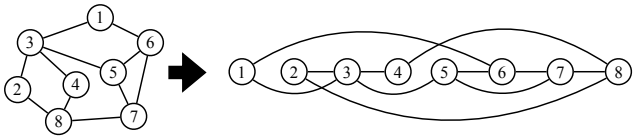

Figure 1 An example of a degeneracy ordering for a 2-degenerate graph G. In this ordering, each vertex v is adjacent to vertices at most two whose indices are larger than v.

we first define the parent-child relationship between solutions as follows: Let $G = (V, E)$ be an input graph with $V = \{v_1, \ldots, v_n\}$ and X and Y be dominating sets on G. We arbitrarily number the vertices in G from 1 to n and call the number of a vertex the *index* of the vertex. If no confusion occurs, we identify a vertex with its index. We assume that there is a total ordering $<$ on V according to the indices. $pv(X)$, called the *parent vertex*, is the vertex in $V \setminus X$ with the minimum index. For any dominating set X such that $X \neq V$, Y is the *parent* of X if $Y = X \cup \{pv(X)\}$. We denote by $\mathcal{P}(X)$ the parent of X. Note that since any superset of a dominating set also dominates G, thus, $\mathcal{P}(X)$ is also a dominating set of G. We call X is a *child* of Y if $\mathcal{P}(X) = Y$. We denote by $\mathcal{F}(G)$ a digraph on the set of solutions $\mathcal{S}(G)$. Here, the vertex set of $\mathcal{F}(G)$ is $\mathcal{S}(G)$ and the edge set $\mathcal{E}(G)$ of $\mathcal{F}(G)$ is defined according to the parent-child relationship. We call $\mathcal{F}(G)$ the *family tree* for G and call V the *root* of $\mathcal{F}(G)$. Next, we show that $\mathcal{F}(G)$ forms a tree rooted at V.

Our basic algorithm `EDS` is shown in Algorithm 1. We say $C(X)$ the *candidate set* of X and define $C(X) = \{v \in V \mid N[X \setminus \{v\}] = V \wedge \mathcal{P}(X \setminus \{v\}) = X\}$. Intuitively, the candidate set of X is the set of vertices such that any vertex v in the set, removing v from X generates another dominating set. We show a recursive procedure `AllChildren`$(X, C(X), G)$ actually generates all children of X on $\mathcal{F}(G)$. We denote by $ch(X)$ the set of children of X, and by $gch(X)$ the set of grandchildren of X.

From Lemmas 1, 2, and 3, we can obtain the correctness of `EDS`.

▶ **Lemma 1.** *For any dominating set X, by recursively applying the parent function $\mathcal{P}(\cdot)$ to X at most n times, we obtain V.*

▶ **Lemma 2.** *$\mathcal{F}(G)$ forms a tree.*

▶ **Lemma 3.** *Let X and Y be distinct dominating sets in a graph G. $Y \in ch(X)$ if and only if there is a vertex $v \in C(X)$ such that $X = Y \cup \{v\}$.*

▶ **Theorem 4.** *By traversing $\mathcal{F}(G)$,* `EDS` *solves Problem 1.*

Algorithm 2: EDS-D enumerates all dominating sets in $O(k)$ time per solution.

```
1  Procedure EDS-D(G = (V, E))                                   // G: an input graph
2      for v ∈ V do D_v ← ∅;
3      AllChildren(V, V, 𝒟(V) := {D_1, ..., D_|V|});
4  Procedure AllChildren(X, C, 𝒟)
5      Output X;
6      C' ← ∅; 𝒟' ← 𝒟;                                           // 𝒟' := {D'_1, ..., D'_|V|}
7      for v ∈ C do                                              // v has the largest index in C
8          Y ← X \ {v};
9          C ← C \ {v};                                          // Remove vertices in Del_3(X, v).
10         C(Y) ← Cand-D(X, v, C);   // Vertices larger than v are not in C.
11         𝒟(Y) ← DomList(v, Y, X, C(Y), C' ⊕ C(Y), 𝒟');
12         AllChildren(Y, C(Y), 𝒟(Y));
13         C' ← C(Y); 𝒟' ← 𝒟(Y);
14         for u ∈ N(v)^{v<} do D'_u ← D'_u ∪ {v} ;
15 Procedure Cand-D(X, v, C)
16     Y ← X \ {v}; Del_1 ← ∅; Del_2 ← ∅;
17     for u ∈ (N(v) ∩ C) ∪ N(v)^{v<} do
18         if u < v then
19             if N(u)^{u<} ∩ Y = ∅ ∧ N(u)^{<u} ∩ Y = ∅ then Del_1 ← Del_1 ∪ {u} ;
20         else
21             if N[u] ∩ (X \ C) = ∅ ∧ |N[u] ∩ C| = 2 then Del_2 ← Del_2 ∪ (N[u] ∩ C) ;
22     return C \ (Del_1 ∪ Del_2);                               // C is C(X \ {v})
23 Procedure DomList (v, Y, X, C' ⊕ C(Y), 𝒟')
24     for u ∈ C' ⊕ C(Y) do
25         for w ∈ N(u)^{u<} do
26             if u ∉ D'_w(X) then
27                 if u ∉ C' then D'_w ← D'_w ∪ {u} ;
28                 else D'_w ← D'_w \ {u} ;
29     for u ∈ N(v)^{v<} do
30         if u ∈ X then D'_v ← D'_v ∪ {u} ;
31     return 𝒟';                                                // 𝒟' is 𝒟(Y)
```

3 Efficient Enumeration for Bounded Degenerate Graphs

The bottle-neck of EDS is the maintenance of candidate sets. Let X be a dominating set and Y be a child of X. We can easily see that the time complexity of EDS is $O(\Delta^2)$ time per solution since a removed vertex $u \in C(X) \setminus C(Y)$ has the distance at most two from v. In this section, we improve EDS by focusing on the degeneracy of an input graph G. G is a *k-degenerate graph* [22] if for any induced subgraph H of G, the minimum degree in H is less than or equal to k. The *degeneracy* of G is the smallest k such that G is k-degenerate. A k-degenerate graph has a *good* vertex ordering. The definition of orderings of vertices in G, called a *degeneracy ordering* of G, is as follows: for any vertex v in G, the number of vertices that are larger than v and adjacent to v is at most k. We show an example of a degeneracy ordering of a graph in Fig. 1. Matula and Beck show that the degeneracy and a degeneracy ordering of G can be obtained in $O(n+m)$ time [24]. Our proposed algorithm

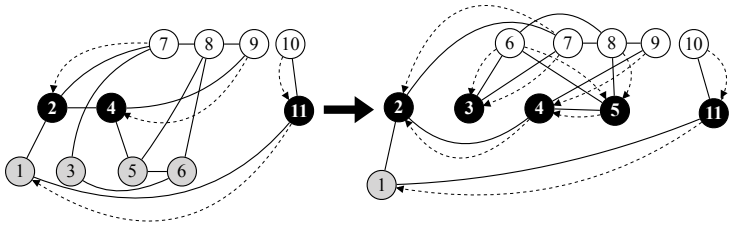

Figure 2 Let X be a dominating set $\{1, 2, 3, 4, 5, 6, 11\}$. An example of the maintenance of $C(X)$ and $\mathcal{D}(X)$. Each dashed directed edge is stored in $\mathcal{D}(X)$, and each solid edge is an edge in G. A directed edge (u, v) implies $v \in D_u(X)$. The index of each vertex is according to a degeneracy ordering. White, black, and gray vertices belong to $V \setminus X$, $X \setminus C(X)$, and $C(X)$, respectively. When `EDS-D` removes vertex 6, $C(X \setminus \{6\}) = \{1\}$.

`EDS-D`, shown in Algorithm 2, achieves amortized $O(k)$ time enumeration by using this good ordering. In what follows, we fix some degeneracy ordering of G and number the indices of vertices from 1 to n according to the degeneracy ordering. We assume that for each vertex v and each dominating set X, $N[v]$ and $C(X)$ are stored in a doubly linked list and sorted by the ordering. Note that the larger neighbors of v can be listed in $O(k)$ time. Let us denote by $V^{<v} = \{1, 2, \ldots, v-1\}$ and $V^{v<} = \{v+1, \ldots n\}$. Moreover, $A^{<v} = A \cap V^{v<}$ and $A^{v<} = A \cap V^{<v}$ for a subset A of V. We first show the relation between $C(X)$ and $C(Y)$.

▶ **Lemma 5.** *Let X be a dominating set of G and Y be a child of X. Then, $C(Y) \subset C(X)$.*

From the Lemma 5, for any $v \in C(X)$, what we need to obtain the candidate set of Y is to compute $Del(X, pv(Y)) = C(X) \setminus C(Y)$, where $Y = X \setminus \{v\}$. In addition, we can easily sort $C(Y)$ by the degeneracy ordering if $C(X)$ is sorted. In what follows, we denote by $Del_1(X, v) = \{u \in C(X)^{<v} \mid N[u] \cap X = \{u, v\}\}$, $Del_2(X, v) = \{u \in C(X)^{<v} \mid \exists w \in V \setminus (X \setminus \{v\})(N[w] \cap X = \{u, v\})\}$, and $Del_3(X, v) = C(X)^{v\leq}$. Next, we show the time complexity for obtaining $Del(X, pv(Y))$.

▶ **Lemma 6.** *For each $v \in C(X)$, $Del(X, v) = Del_1(X, v) \cup Del_2(X, v) \cup Del_3(X, v)$ holds.*

We show an example of dominated list and a maintenance of $C(X)$ in Fig. 2. To compute a candidate set efficiently, for each vertex u in V, we maintain the vertex lists $D_u(X)$ for X. We call $D_u(X)$ the *dominated list* of u for X. The definition of $D_u(X)$ is as follows: If $u \in V \setminus X$, then $D_u(X) = N(u) \cap (X \setminus C(X))$. If $u \in X$, then $D_u(X) = N(u)^{<u} \cap (X \setminus C(X))$. For brevity, we write D_u as $D_u(X)$ if no confusion arises. We denote by $\mathcal{D}(X) = \bigcup_{u \in V} \{D_u\}$. By using $\mathcal{D}(X)$, we can efficiently find $Del_1(X, v)$ and $Del_2(X, v)$.

▶ **Lemma 7.** *Let X be a dominating set of G. Suppose that for each vertex u in G, we can obtain the size of D_u in constant time. Then, for each vertex $v \in C(X)$, we can compute $Del_1(X, v)$ in $O(k)$ time on average over all children of X.*

▶ **Lemma 8.** *Suppose that for each vertex w in G, we can obtain the size of D_w in constant time. For each vertex $v \in C(X)$, we can compute $Del_2(X, v)$ in $O(k)$ time on average over all children of X.*

In Lemma 7 and Lemma 8, we assume that the dominated lists were computed when we compute $Del(X, v)$ for each vertex v in $C(X)$. We next consider how we maintain $\mathcal{D}$. Next lemmas show the transformation from $D_u(X)$ to $D_u(Y)$ for each vertex u in G.

▶ **Lemma 9.** *Let X be a dominating set, v be a vertex in $C(X)$, and $Y = X \setminus \{v\}$. For each vertex $u \in G$ such that $u \neq v$, $D_u(Y) = D_u(X) \cup (N(u)^{<u} \cap (Del_1(X,v) \cup Del_2(X,v))) \cup (N(u)^{<u} \cap (Del_3(X,v) \setminus \{v\}))$.*

▶ **Lemma 10.** *Let X be a dominating set, v be a vertex in $C(X)$, and $Y = X \setminus \{v\}$. $D_v(Y) = D_v(X) \cup (N(v)^{<v} \cap (Del_1(X,v) \cup Del_2(X,v))) \cup (N(v)^{v<} \cap X)$.*

We next consider the time complexity for obtaining the dominated lists for children of X. From Lemma 9 and Lemma 10, a naïve method for the computation needs $O(k|Del(X,v)|+k)$ time for each vertex v of X since we can list all larger neighbors of any vertex in $O(k)$ time. However, if we already know $C(W)$ and $\mathcal{D}(W)$ for a child W of X, then we can easily obtain $\mathcal{D}(Y)$, where Y is the child of X immediately after W. The next lemma plays a key role in `EDS-D`. Here, for any two sets A, B, we denote by $A \oplus B = (A \setminus B) \cup (B \setminus A)$.

▶ **Lemma 11.** *Let X be a dominating set, v, u be vertices in $C(X)$ such that u has the maximum index in $C(X)^{<v}$, $Y = X \setminus \{u\}$, and $W = X \setminus \{v\}$. Suppose that we already know $C(Y) \oplus C(W)$, $\mathcal{D}(W)$, $Del(X,v)$, and $Del(X,u)$. Then, we can compute $\mathcal{D}(Y)$ in $O(k|C(Y) \oplus C(W)| + k)$ time.*

Proof. Suppose that z is a vertex in G such that $z \neq v$ and $z \neq u$. From the definition, $D_z(W) \setminus D_z(Y) = (Del(X,v) \setminus Del(X,u)) \cap N(z)^{<z}$ and $D_z(Y) \setminus D_z(W) = (Del(X,u) \setminus Del(X,v)) \cap N(z)^{<z}$. Hence, we first compute $Del(X,v) \oplus Del(X,u)$. Now, $(C(X) \setminus C(W)) \oplus (C(X) \setminus C(Y)) = C(W) \oplus C(Y)$. Next, for each vertex c in $C(W) \oplus C(Y)$, we check whether we add to or remove c from $D_z(Y)$ or not. Note that added or removed vertices from $D_z(Y)$ is a smaller neighbor of z. From the definition, if $c \notin D_z(Y)$ or $c \in D_z(X)$, then we add c to $D_z(Y)$. Otherwise, we remove c from $D_z(Y)$. Thus, since each vertex in $C(W) \oplus C(Y)$ has at most k larger neighbors, for all vertices other than u and v, we can compute the all dominated lists in $O(k|C(W) \oplus C(Y)|)$ time. Next we consider the update for $D_u(Y)$ and $D_v(Y)$. Note that from the definition, $D_v(W)$ and $D_u(Y)$ contain larger neighbors of v and u, respectively. However, the number of such neighbors is $O(k)$. Finally, since v belongs to Y, $v \in D_{u'}(Z)$ if $u' \in N(v)^{v<}$ for any vertex u'. Thus, as with the above discussion, we can compute $D_u(Y)$ and $D_v(Y)$ in $O(k|C(W) \oplus C(Y)| + k)$ time. ◀

▶ **Lemma 12.** *Let X be a dominating set. Then, `AllChildren`$(X, C(X), \mathcal{D}(X))$ of `EDS-D` other than recursive calls can be done in $O(k|ch(X)| + k|gch(X)|)$ time.*

Proof. We first consider the time complexity of `Cand-D`. From Lemma 7 and Lemma 8, `Cand-D` correctly computes $Del_1(X,v)$ and $Del_2(X,v)$ in from line 18 to line 19 and from line 20 to line 21, respectively. For each loop from line 7, the algorithm picks the largest vertex in C. This can be done in $O(1)$ since C is sorted. The algorithm needs to remove vertices in $Del_3(X,v)$. This can be done in line 9 and in $O(1)$ time since v is the largest vertex. Thus, for each vertex v in $C(X)$, $C(X \setminus \{v\})$ can be obtained in $O(k)$ time on average. Hence, for all vertices in $C(X)$, the candidate sets can be computed in $O(k|ch(X)|)$ time. Next, we consider the time complexity of `DomList`. Before computing `DomList`, `EDS-D` already computed $C(Y) \oplus C(W)$, $\mathcal{D}(W)$, $Del(X,v)$, and $Del(X,v')$. Note that we can compute $C(Y) \oplus C(W)$ when we compute $C(Y)$ and $C(W)$. Here, W is the previous dominating set, C' stores $C(W)$, and $\mathcal{D}'$ stores $\mathcal{D}(W)$. Thus, by using Lemma 11, we can compute $\mathcal{D}(Y)$ in $O(k|C(Y) \oplus C(W)| + k)$ time. In addition, for all vertices in $C(X)$, the dominated lists can be computed in $O(k|C(X)| + k|gch(X)|)$ time since Y has at least $|C(W) \setminus C(Y)| - 1$ children and $|gch(X)|$ is at least the sum of $|C(W) \setminus C(Y)| - 1$ over all $Y \in \{X \setminus \{v\} \mid v \in C(X)\}$ and the previous solution W of Y. When `EDS-D` copies data

such as $\mathcal{D}$, EDS-D only copies the pointer of these data. By recording operations of each line, EDS-D restores these data when backtracking happens. These restoring can be done in the same time of the above update computation. ◀

▶ **Theorem 13.** EDS-D *enumerates all dominating sets in* $O(k)$ *time per solution in a* k*-degenerate graph by using* $O(n+m)$ *space.*

Proof. The parent-child relation of EDS-D and EDS are same. From Lemma 5 and Lemma 6, EDS-D correctly computes all children. Hence, the correctness of EDS-D is shown by the same manner of Theorem 4. We next consider the space complexity of EDS-D. For any vertex v in G, if v is removed from a data structure used in EDS-D on a recursive procedure, v will never be added to the data structure on descendant recursive procedures. In addition, for each recursive procedure, the number of data structures that are used in the procedure is constant. Hence, the space complexity of EDS-D is $O(n+m)$. We finally consider the time complexity. Each recursive procedure needs $O(k|ch(X)| + k|gch(X)|)$ time from Lemma 12. Thus, the time complexity of EDS-D is $O\left(k\sum_{X\in\mathcal{S}}(|ch(X)| + |gch(X)|)\right)$, where $\mathcal{S}$ is the set of solutions. Now, $O\left(\sum_{X\in\mathcal{S}}(|ch(X)| + |gch(X)|)\right) = O(|\mathcal{S}|)$. Hence, the statement holds. ◀

4 Efficient Enumeration for Graphs with Girth at Least Nine

In this section, we propose an optimum enumeration algorithm EDS-G for graphs with girth at least nine, where the girth of a graph is the length of a shortest cycle in the graph. That is, the proposed algorithm runs in constant amortized time per solution for such graphs. The algorithm is shown in Algorithm 3. To achieve constant amortized time enumeration, we focus on the *local structure* $G_v(X)$ for (X, v) of G defined as follows: $G_v(X) = G[(V \setminus N[X \setminus C(X)^{\leq v}]) \cup C(X)^{\leq v}]$. Fig. 3 shows an example of $G_v(X)$. $G_v(X)$ is a subgraph of G induced by vertices that (1) are dominated by vertices only in $C(X)^{\leq v}$ or (2) are in $C(X)^{\leq v}$. Intuitively speaking, we can efficiently enumerate solutions by using the local structure and ignoring vertices in $G \setminus G_v(X)$ since the number of solutions that are generated according to the structure is enough to reduce the *amortized* time complexity to constant. We denote by $G(X) = G[(V \setminus N[X \setminus C(X)]) \cup C(X)]$ the local structure for (X, v_*) of G, where v_* is the largest vertex in G.

We first consider the correctness of EDS-G. The parent-child relation between solutions used in EDS-G is the same as in EDS. Suppose that X and Y are dominating sets such that X is the parent of Y. Recall that, from Lemma 6, $C(X) \setminus C(Y) = Del(X, v)$, where $X = Y \cup \{v\}$. We denote by $f_v(u, X) =$ True if there exists a neighbor w of u such that $w \in X \setminus C(X)^{\leq v}$; Otherwise $f_v(u, X) =$ False. Thus, Cand-G correctly computes $Del_1(X, v)$ and $Del_2(X, v)$ from line 17 to 19. Moreover, in line 14, vertices in $Del_3(X, v)$ are removed from $C(X)$ and hence, Cand-G also correctly computes $C(X \setminus \{v\})$. Moreover, for each vertex w removed from G during enumeration, w is dominated by some vertices in G. Hence, by the same discussion as Theorem 4, we can show that EDS-G enumerates all dominating sets. In the remaining of this section, we show the time complexity of EDS-G. Note that $G_v(X)$ does not include any vertex in $N[Del_3(X, v) \setminus \{v\}] \setminus C(X)^{\leq v}$. Hence, we will consider only vertices in $Del_1(X, v) \cup Del_2(X, v) \cup \{v\}$. We denote by $Del'(X, v) = Del_1(X, v) \cup Del_2(X, v) \cup \{v\}$. We first show the time complexity for updating the candidate sets.

In what follows, if v is the largest vertex in $C(X)$, then we simply write $f(u, X)$ as $f_v(u, X)$. We denote by $N'_v(u) = N_{G_v(X)}(u)$, $N'_v[u] = N'_v(u) \cup \{u\}$, and $d'_v(u) = |N'_v(u)|$ if no confusion arises. Suppose that G and $G_v(X)$ are stored in an adjacency list, and neighbors of a vertex are stored in a doubly linked list and sorted in the ordering.

Algorithm 3: `EDS-G` enumerates all dominating sets in $O(1)$ time per solution for a graph with girth at least nine.

```
1  Procedure EDS-G(G = (V, E))                                  // G: an input graph
2      for v ∈ V do f_v ← False ;
3      AllChildren (V, V, {f_1, ..., f_|V|}, G);
4  Procedure AllChildren (X, C, F, G)
5      Output X;
6      for v ∈ C(X) do                                  // v is the largest vertex in C
7          Y ← X \ {v};
8          (C(Y), F(Y), G(Y)) ← Cand-G (v, C, F, G);
9          AllChildren (Y, C(Y), F(Y), G(Y));
10         for u ∈ N_G(v) do
11             if u ∈ C then f_u ← True ;
12             else G ← G \ {u} ;
13         G ← G \ {v};
14         C ← C \ {v};                          // Remove vertices in Del_3(X, v).
15 Procedure Cand-G (v, C, F, G)
16     Del_1 ← ∅; Del_2 ← ∅;
17     for u ∈ N_G(v) do
18         if N_G[u] ∩ X = {u, v} and f_u = False then Del_1 ← Del_1 ∪ {u} ;
19         else if ∃w(N_G[u] ∩ X = {w, v}) then Del_2 ← Del_2 ∪ {w} ;
20     C' ← C \ (Del_1 ∪ Del_2 ∪ {v});
21     for u ∈ N'[Del_1 ∪ Del_2] do                                   // Lemma 17
22         f_u ← True;
23         if u ∉ C' then G ← G \ {u} ;
24     if f_v = True then G ← G \ {v};
25     return (C', F, G);
```

▶ **Lemma 14.** *Let X be a dominating set, v be a vertex in $C(X)$, and u be a vertex in G. Then, $u \in Del_1(X, v)$ if and only if $N'_v[u] \cap X = \{u, v\}$ and $f_v(u, X) = \texttt{False}$.*

▶ **Lemma 15.** *Let X be a dominating set, v be a vertex in $C(X)$, and u be a vertex in G. Then, $u \in Del_2(X, v)$ if and only if there is a vertex w in $G_v(X)$ such that $N'_v[w] \cap X = \{u, v\}$.*

▶ **Lemma 16.** *Let X be a dominating set and v be a vertex in $C(X)$. Suppose that for any vertex u, we can check the number of u's neighbors in the local structure $G_v(X)$ and the value of $f_v(u, X)$ in constant time. Then, we can compute $C(X \setminus \{v\})$ from $C(X)^{\leq v}$ in $O(d'_v(v))$ time*

▶ **Lemma 17.** *Let X be a dominating set, v be a vertex in $C(X)$, and $Y = X \setminus \{v\}$. Then, we can compute $G(Y)$ from $G_v(X)$ in $O\left(\sum_{u \in Del'(X,v)} d'_v(u) + \sum_{u \in G_v(X) \setminus G(Y)} d'_v(u)\right)$ time. Note that $N'_v(u) = N_{G_v(X)}(u)$ and $d'_v(u) = |N'_v(u)|$.*

From Lemma 16 and Lemma 17, we can compute the local structure and the candidate set of Y from those of X in $O\left(\sum_{u \in Del'(X,v)} d'_v(u) + \sum_{u \in G_v(X) \setminus G(Y)} d'_v(u)\right)$ time. We next consider the time complexity of the loop in line 10. In this loop procedure, `EDS-G` deletes all the neighbors u of v from $G_v(X)$ if $u \notin C(X)^{\leq v}$ because for each descendant W of dominating set Y', $v \in W \setminus C(W)$, where Y' is a child of X and is generated after Y. Thus,

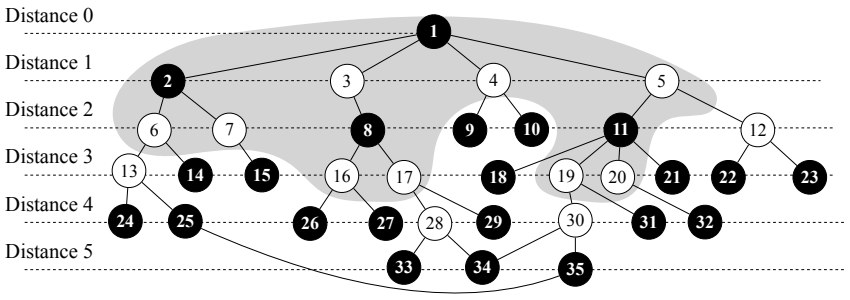

Figure 3 An example of $G_v(X)$, where $v = 1$. The vertices in the grey area are $Del'(X, v) \cup (G_v(X) \setminus G(Y)) \cup (N'_v(v) \setminus X)$. Each horizontal line represents the distance between 1 and any vertex.

this needs $O\left(d'_v(v) + \sum_{u \in N'(v) \setminus X} d'_v(u)\right)$ time. Hence, from the above discussion, we can obtain the following lemma:

▶ **Lemma 18.** *Let X be a dominating set, v be a vertex in $C(X)$, and $Y = X \setminus \{v\}$. Then,* `AllChildren` *other than a recursive call runs in the following time bound:*

$$O\left(\sum_{u \in Del'(X,v)} d'_v(u) + \sum_{u \in G_v(X) \setminus G(Y)} d'_v(u) + \sum_{u \in N'_v(v) \setminus X} d'_v(u)\right). \tag{1}$$

Before we analyze the number of descendants of X, we show the following lemmas.

▶ **Lemma 19.** *Let us denote by $Pen_v(X) = \{u \in Del'(X, v) \mid d'_v(u) = 1\}$. Then, $\sum_{v \in C(X)} |Pen_v(X)|$ is at most $|C(X)|$.*

Let v be a vertex in $C(X)$ and a pendant in $G_v(X)$. Since the number of such pendants is at most $|C(X)|$, the sum of degree of such pendants is at most $|C(X)|$ in each execution of `AllChildren` without recursive calls. Hence, the cost of deleting such pendants is $O(|C(X)|)$ time. Next, we consider the number of descendants of X. From Lemma 19, we can ignore such pendant vertices. Hence, for each $u \in Del'(X, v)$, we will assume that $d'_v(u) \geq 2$ below.

▶ **Lemma 20.** *Let X be a dominating set, v be a vertex in $C(X)$, and Y be a dominating set $X \setminus \{v\}$. Then, $|C(Y)|$ is at least $|(N'_v(v) \cap X) \setminus Del'(X, v)|$.*

▶ **Lemma 21.** *Let X be a dominating set, v be a vertex in $C(X)$, and Y be a dominating set $X \setminus \{v\}$. Then, $|C(Y)|$ is at least $\sum_{u \in N'_v(v) \setminus X} (d'_v(u) - 1)$.*

▶ **Lemma 22.** *Let X be a dominating set, v be a vertex in $C(X)$, and Y be a dominating set $X \setminus \{v\}$. Then, $|C(Y)|$ is at least $\sum_{u \in Del'(X,v) \setminus \{v\}} (d'_v(u) - 1)$.*

▶ **Lemma 23.** *Let X be a dominating set v be a vertex in $C(X)$, and Y be a dominating set $X \setminus \{v\}$. Then, the number of children and grandchildren of Y is at least $\sum_{u \in G_v(X) \setminus (G(Y) \cup Del'(X,v) \cup N'_v(v))} (d'_v(u) - 1)$.*

Note that for any pair of candidate vertices v and v', $X \setminus \{v\}$ and $X \setminus \{v'\}$ do not share their descendants. Thus, from Lemma 20, Lemma 21, Lemma 22, and Lemma 23, we can obtain the following lemma:

▶ **Lemma 24.** *Let X be a dominating set. Then, the sum of the number of X's children, grandchildren, and great-grandchildren is bounded by the following order:*

$$\Omega\left(|C(X)| + \sum_{v\in C(X)}\left(\sum_{u\in Del'(X,v)} d'_v(u) + \sum_{u\in G_v(X)\setminus G(Y)} d'_v(u) + \sum_{u\in N'_v(v)\setminus X} d'_v(u)\right)\right). \quad (2)$$

From Lemma 18, Lemma 19, and Lemma 24, each iteration outputs a solution in constant amortized time. Hence, by the same discussion of Theorem 13, we can obtain the following theorem.

▶ **Theorem 25.** *For an input graph with girth at least nine,* `EDS-G` *enumerates all dominating sets in $O(1)$ time per solution by using $O(n+m)$ space.*

Proof. The correctness of `EDS-G` is shown by Theorem 4, Lemma 14, and Lemma 15. By the same discussion with Theorem 13, the space complexity of `EDS-G` is $O(n+m)$. We next consider the time complexity of `EDS-G`. From Lemma 18, Lemma 19, and Lemma 24. we can amortize the cost of each recursion by distributing $O(1)$ time cost to the corresponding descendant discussed in the above lemmas. Thus, the amortized time complexity of each recursion becomes $O(1)$. Moreover, each recursion outputs a solution. Hence, `EDS-G` enumerates all solutions in $O(1)$ amortized time per solution. ◀

5 Conclusion

In this paper, we proposed two enumeration algorithms. `EDS-D` solves the dominating set enumeration problem in $O(k)$ time per solution by using $O(n+m)$ space, where k is a degeneracy of an input graph G. Moreover, `EDS-G` solves this problem in constant time per solution if an input graph has girth at least nine.

Our future work includes to develop efficient dominating set enumeration algorithms for dense graphs. If a graph is dense, then k is large and G has many dominating sets. For example, in the case of complete graphs, k is equal to $n-1$ and every nonempty subset of V is a dominating set. That is, the number of solutions for a dense graph is much larger than that for a sparse graph. This allows us to spend more time in each recursive call. However, `EDS-D` is not efficient for dense graphs although the number of solutions is large. Moreover, if G is small girth, that is, G is dense then `EDS-G` does not achieve constant amortized time enumeration. Hence, the dominating set enumeration problem for dense graphs is interesting.

References

1 David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Appl. Math.*, 65(1):21–46, 1996.

2 Etienne Birmelé, Rui A. Ferreira, Roberto Grossi, Andrea Marino, Nadia Pisanti, Romeo Rizzi, and Gustavo Sacomoto. Optimal Listing of Cycles and st-Paths in Undirected Graphs. In *Proc. SODA 2013 ACM*, pages 1884–1896, 2013.

3 Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. Generating maximal independent sets for hypergraphs with bounded edge-intersections. In *Proc. LATIN 2004*, pages 488–498. Springer, 2004.

4 Boštjan Brešar, Michael A Henning, and Douglas F Rall. RAINBOW DOMINATION IN GRAPHS. *Taiwanese J. Math.*, 12(1):213–225, 2008.

5 L Sunil Chandran and CR Subramanian. Girth and treewidth. *J. Combin. Theory Ser. B*, 93(1):23–32, 2005.

6 Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. Sublinear-Space Bounded-Delay Enumeration for Massive Network Analytics: Maximal Cliques. In *Proc. ICALP 2016*, volume 55 of *LIPIcs*, pages 148:1–148:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.148.

7 Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Appl. Math.*, 157(12):2675–2700, 2009.

8 Nadia Creignou, Markus Kröll, Reinhard Pichler, Sebastian Skritek, and Heribert Vollmer. On the Complexity of Hard Enumeration Problems. In *Proc. LATA 2017*, volume 10168 of *LNCS*, pages 183–195. Springer, 2017.

9 Jean-Alexandre Anglès d'Auriac, Csilia Bujtás, Hakim El Maftouhi, Marek Karpinski, Yannis Manoussakis, Leandro Montero, Narayanan Narayanan, Laurent Rosaz, Johan Thapper, and Zsolt Tuza. Tropical Dominating Sets in Vertex-Coloured Graphs. In *Proc. WALCOM 2016*, pages 17–27. Springer, 2016.

10 Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New Results on Monotone Dualization and Generating Hypergraph Transversals. *SIAM J. Comput.*, 32(2):514–537, 2003. doi:10.1137/S009753970240639X.

11 David Eppstein, Maarten Löffler, and Darren Strash. Listing All Maximal Cliques in Large Sparse Real-World Graphs. *J. Exp. Algorithmics*, 18:3.1:3.1–3.1:3.21, November 2013. doi:10.1145/2543629.

12 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

13 Petr A Golovach, Pinar Heggernes, Mamadou M Kanté, Dieter Kratsch, and Yngve Villanger. Enumerating minimal dominating sets in chordal bipartite graphs. *Discrete Appl. Math.*, 199(30):30–36, 2016.

14 Petr A Golovach, Pinar Heggernes, Mamadou Moustapha Kanté, Dieter Kratsch, Sigve H Sæther, and Yngve Villanger. Output-Polynomial Enumeration on Graphs of Bounded (Local) Linear MIM-Width. *Algorithmica*, 80(2):714–741, 2018.

15 Petr A. Golovach, Pinar Heggernes, Dieter Kratsch, and Yngve Villanger. An Incremental Polynomial Time Algorithm to Enumerate All Minimal Edge Dominating Sets. *Algorithmica*, 72(3):836–859, 2015. doi:10.1007/s00453-014-9875-7.

16 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. Enumeration of Minimal Dominating Sets and Variants. In *Proc. FCT 2011*, pages 298–309. Springer, 2011.

17 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the Neighbourhood Helly of Some Graph Classes and Applications to the Enumeration of Minimal Dominating Sets. In *Proc. ISAAC 2012*, volume 7676, pages 289–298. Springer, 2012.

18 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the Enumeration of Minimal Dominating Sets and Related Notions. *SIAM J. Discrete Math.*, 28(4):1916–1929, 2014.

19 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, Lhouari Nourine, and Takeaki Uno. A polynomial delay algorithm for enumerating minimal dominating sets in chordal graphs. In *Proc. WG 2015*, pages 138–153. Springer, 2015.

20 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, Lhouari Nourine, and Takeaki Uno. Polynomial Delay Algorithm for Listing Minimal Edge Dominating Sets in Graphs. In *Proc. WADS 2015*, volume 9214 of *LNCS*, pages 446–457. Springer Berlin Heidelberg, 2015.

21 E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Generating All Maximal Independent Sets: NP-Hardness and Polynomial-Time Algorithms. *SIAM J. Comput.*, 9(3):558–565, 1980. doi:10.1137/0209042.

22 Don R Lick and Arthur T White. k-DEGENERATE GRAPHS. *Canadian J. Math.*, 22:1082–1096, 1970.

23 Kazuhisa Makino and Takeaki Uno. New Algorithms for Enumerating All Maximal Cliques. In *Proc. SWAT 2004*, volume 3111 of *LNCS*, pages 260–272. Springer, 2004.

24 David W Matula and Leland L Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, 1983.

25 Akiyoshi Shioura, Akihisa Tamura, and Takeaki Uno. An Optimal Algorithm for Scanning All Spanning Trees of Undirected Graphs. *SIAM J. Comput.*, 26(3):678–692, 1997.

26 Andrew Thomason. The Extremal Function for Complete Minors. *Journal of Combinatorial Theory, Series B*, 81(2):318–338, 2001.

27 Kunihiro Wasa, Hiroki Arimura, and Takeaki Uno. Efficient Enumeration of Induced Subtrees in a K-Degenerate Graph. In *Proc. ISAAC 2014*, volume 8889 of *LNCS*, pages 94–102. Springer, 2014. doi:10.1007/978-3-319-13075-0_8.