Efficient Enumeration of Substructures in Sparse Graphs

（疎なグラフにおける部分構造の効率良い列挙）

Kazuhiro Kurita

February 2020

Division of Computer Science and Information Technology

Graduate School of Information Science and Technology

Hokkaido University

Abstract

Graphs are widely used to represent various data. For example, communication networks, metabolic networks, and social networks are graphs in the real world. In this thesis, we address efficient enumeration for sparse graphs. The first parameter of sparsity is girth. Small cycles make a problem difficult not only an enumeration problem but also optimization problems. For example, the minimum dominating set, the maximum independent set, and the maximum induced matching problem have a fixed parameter tractability. Indeed, we develop efficient enumeration algorithms for dominating sets and induced matchings if an input graph has large girth. In addition, we address another approach that uses the sparsity for subgraph enumeration. We next consider the problem which the output has girth constraint. In addition to girth, the other parameters of sparsity are degeneracy and degree. We develop several theoretical efficient enumeration algorithms. To show that these algorithms are practical, we experiment these algorithms for artificial graph data. As a result, our algorithms are faster than simple algorithms. In what follows, we explain our main results.

In Chapter 3, we use girth as the sparsity. More precisely, we assume that an input graph has no cycles with length four. We address an induced matching enumeration problem. An induced matching is a set of edges such that the distance between any pair of edges is at least two. We propose a binary partition based enumeration algorithm.

Binary partition is one of the most famous technique to construct an enumeration algorithm. Our algorithm runs in constant amortized time. In addition, our algorithm is 200 times faster than a simple algorithm.

In Chapter 4, we use girth once again as the sparsity. However, in this chapter, we use girth constraint to output. We address enumeration of subgraphs and induced subgraphs with bounded girth. We consider directed case and undirected case. For all cases, we achieves $O(n)$ delay enumeration, where n is the number of vertices. The bottleneck of this algorithm is girth computation. To speed up girth computation, these algorithms use a kind of distance matrix. We implement one algorithm and it is 60 times faster than a simple algorithm.

In Chapter 5, we introduce the new sparsity parameter, *degeneracy*. We consider enumeration of dominating sets and propose two algorithms. These algorithms run in amortized $O(k)$ time and constant amortized time, respectively, where k is the degeneracy of a graph. It is known that a k-degenerate graph has a good vertex elimination ordering. This ordering is important in the former algorithm. The latter algorithm needs $O(\Delta^3)$ time in each node, where Δ is the degree of a graph. However, the number of children and grandchildren is enough large if an input graph has the large girth. Hence, the amortized time complexity improves from $O(\Delta^3)$ time to $O(1)$ time.

In Chapter 6, we consider enumeration of chordal bipartite induced subgraphs. The recognition of a dominating set and an induced matching is very simple. It is sufficient to check the neighborhood of each element. However, recognition of chordal bipartite graphs is non trivial. To solve this problem, the bottleneck is recognition of chordal bipartite graphs. Hence, we develop a new characterization of chordal bipartite graphs which convenient for enumeration. We show that chordal bipartite graphs have vertex

elimination ordering. Based on this ordering, we develop an enumeration algorithm which runs in $O(kt\Delta^2)$ time, where t is the size of a maximum biclique $K_{t,t}$ of a graph.

Finally, we summarize our results in Chapter 7. In addition, we give future directions of this thesis.

Acknowledgement

I would like to express my special appreciation and thanks to my supervisor Hiroki Arimura. This thesis would not have been possible without his support and encouragement. I would also like to express thanks to associate professor Takuya Kida. He always helped me when I was in trouble and support my life in Hokkaido University.

I would like to express my gratitude to Takeaki Uno and Kunihiro Wasa for supporting a big part of this work. I am grateful to Roberto Grossi for his hosting me in his laboratory. I wolud like to thank to Alessio Conte, Andrea Marino, and Giulia Punzi for helpful discussion.

Ms. Manabe always helped us with our daily life in our laboratory. Last but not least, my deepest gratitude to my parents Yasushi and Kaoru, my brothers Takafumi and Yohei, and all my family. They always supported me for 27 years.

Contents

Chapter 1

Introduction

1.1 Background

Graphs represent various real world data, i.e., road networks, social networks, and citation networks. Dense subgraph enumeration has been studied in order to analyze such graphs [12, 20]. A graph is called *dense* if the number of edges is large. For example, a clique, that is a subgraph which any pair of vertices are adjacent, is a typical dense subgraph. The sparsity is often used to develop efficient enumeration algorithm for maximal cliques [9, 10, 15, 21].

In this thesis, we develop efficient enumeration algorithms for sparse graphs. A graph is called *sparse* if the number of edges is small. First, we consider graphs with no short cycles as sparse graphs. More precisely, we consider graphs with *girth* at least g. The girth is the length of a shortest cycle in a graph. The reason for this definition is that many small cycles make several graph optimization problems difficult. Indeed, the maximum independent set problem and the t-vertex cover problem are tractable if a graph has the large girth [59]. If the girth of a graph is large, then a set of vertices

such that any pair of vertices has the distance at most $g/2$ induce a tree. A tree has $n-1$ edges, where n is the number of vertices. Hence, a tree is sparse graphs. From this observation, we adopt girth as the sparsity parameter. Next, there are two approaches to using the sparsity for subgraph enumeration. One is that input graphs have large girth constraint. The other is that outputs have large girth constraint. We address both problems. In addition to the theoretical analysis, we consider whether our algorithm can be applied to data analysis. Hence, we implement and show the performance of our algorithms.

1.2 Research Goal

The aim of this thesis is to develop an efficient enumeration algorithm for sparse graphs. The time complexity of enumeration algorithms is measured with respect to the size of input and the number of solutions. An enumeration algorithm is called amortized polynomial time algorithm if it runs in linear time with respect to the number of solutions. Then, the goal is to develop an enumeration algorithm which runs in *constant time per solution (constant amortized time)*. To solve an enumeration problem, a basic technique is backtracking. For each element, we recursively search two cases. One case adds this element to a solution. The other case does not add this element to a solution. See Section 2.3 for the details. However, by applying this technique simply, we cannot achieve constant amortized time. Hence, we use the sparsity.

In addition, we implement the following three algorithms, the induced matching enumeration algorithm `EIM`, the connected subgraph with bounded girth enumeration in undirected graphs `EUG-ES`, and the dominating set enumeration algorithm `EDS-D`. We compare the performance of our algorithms, simple algorithms, and naive algorithms.

As a result, our proposed algorithm `EIM` and `EUG-ES` are faster than simple algorithms. However, `EDS-D` is slower than a naive algorithm. In what follows, we show the main results.

1.3 Enumeration of induced matchings

The maximum induced matching problem is tractable for graphs with girth six or more [54]. We develop an algorithm `EIM` for induced matchings in C_4-free graphs which has no cycle with length four. The algorithm `EIM` runs in constant amortized time. An induced matching is a set of edges such that any pair of edges has the distance at least two. By applying simple binary partition, we can enumerate all induced matchings in amortized $O(\Delta^2)$ time, where Δ is the degree of a graph. To reduce the time complexity, we manage the order of selecting edges and improve the cost from $O(\Delta^2)$ to $O(1)$[1].

In low-degree polynomial time enumeration algorithms, the most time-consuming part is often typically the generation of child problems. If the structure is simple, child problem generation can be completed within a short time. For example, if the graph is a tree, there is no cycle around a vertex; thus, we do not have to consider the unification of multiple edges when we shrink an edge. Not only theoretical analysis of `EIM`, we implement `EIM` and compare the performance of `EIM`, a simple algorithm, and a naive algorithm. In our experiment, `EIM` is 200 times faster than a simple algorithm.

[1] This result is published in [45].

1.4 Enumeration of subgraphs with bounded girth

We address enumeration of sparse graphs in this chapter. In particular, we consider enumeration of subgraphs with bounded girth. The girth is the length of a shortest cycle in a graph. We consider enumeration of subgraphs and induced subgraphs with bounded girth in a directed graph and an undirected graph. This problem generalizes that of the acyclic subgraphs enumeration. If g is more than n, then this problem corresponds to the acyclic subgraph enumeration.

The problem of finding the acyclic subgraph with maximum size or weight has been studied. However, the best of our knowledge, this is the first efficient enumeration algorithm for the problem. Our proposed algorithms have polynomial delay and solve the problem for both directed an undirected graphs. The girth computation is a bottleneck of a simple binary partition algorithm. To overcome this complexity, we use the all to all distance matrix and the *second distance matrix*. The second distance matrix maintains the length of a second shortest path for any pair of vertices. By using these data structures, we achieves $O(n)$ delay enumeration for four enumeration problems[2]. We implement the proposed algorithm for connected subgraphs with large girth. It runs in $O(n)$ time per solution. We show the performance of this algorithm. This algorithm is 60 times faster than a simple algorithm.

1.5 Enumeration of dominating sets

The minimum dominating set problem has a fixed parameter tractability for girth five or more [59]. The sparsity make this problem easy. In Chapter 5, we address enumeration of dominating set in sparse graphs.

[2]These results are published in [16, 44]

A set of vertices D is a *dominating set* if a closed neighbor of D is equal to V. In this chapter, we focus on sparse graphs, e.g., bounded degenerate graphs and graphs with large girth. We develop two constant amortized time enumeration algorithms[3]. One is an amortized $O(k)$ time enumeration algorithm for k-degenerate graphs. The other is an amortized $O(1)$ time enumeration algorithm for graphs with girth at least nine. We implement our algorithm for bounded degeneracy graphs `EDS-D`. We investigate the performance of `EDS-D`. However, `EDS-D` is slower than a naive algorithm and a simple algorithm.

1.6 Enumeration of chordal bipartite induced subgraphs

Each element of a dominating set or an induced matching affects only the neighborhood. However, chordal bipartite induced subgraph affect other than neighborhood. In Chapter 6, we address enumeration of chordal bipartite induced subgraph. Our proposed algorithm runs in constant amortized time for bounded degree graphs. Our algorithm runs in amortized $O(kt\Delta^2)$ time, where k is the degeneracy and t is the size of a maximum biclique $K_{t,t}$[4]. Note that k and t are at most Δ. This algorithm correctly works even if we do not know the exact value of t.

Chordal bipartite graphs are bipartite graphs without induced cycles with length six or more. Chordal bipartite graphs have several equivalent characterizations, and are closely related to strongly chordal graphs and β-acyclic hypergraphs [6, 22, 31, 49, 67]. In particular, a chordal bipartite graph also has a vertex elimination ordering, called

[3]This result is published in [43].
[4]This result is published in [46].

weak elimination ordering (`WEO`) [67]. The bottleneck of this problem is a recognition problem of chordal bipartite graphs. In this thesis, we define a new characterization of chordal bipartite graphs, called *chordal bipartite elimination ordering*. By using this ordering, we construct an enumeration algorithm based on reverse search. In addition, this characterization is useful for an enumeration algorithm since we can recognize chordal bipartite graphs in $O(\Delta^2)$ time.

1.7 Related work

We give several results for constant amortized enumeration algorithms. Note that there exist a good survey of enumeration algorithms by Wasa [74]. In particular, the degeneracy of a graph is often used to construct efficient enumeration algorithms.

1.7.1 For general graphs

It is known that the following problems can be solved in constant amortized time: spanning tree enumeration [62, 73], connected induced subgraph enumeration [73], chordal induced subgraphs enumeration [41], perfect elimination ordering enumeration [8], perfect sequence of chordal graph [52], and matching enumeration [73].

The above algorithms are recursive algorithms. In each recursion, algorithms other than [8] need computation time more than $O(1)$. The key technique is amortized analysis. Especially, push out amortization is very powerful tool to show constant amortized time enumeration. Even if much computation time is needed for each recursive call, we can show that enumeration algorithm runs in constant amortized time if the number of descendants is sufficient large.

1.7.2 For sparse graphs: bounded degenerate graphs

It is known that the following problems can be solved in constant amortized time if degeneracy k is constant: induced subtree enumeration (amortized $O(k)$ time [75]), k'-degenerate induced subgraph enumeration (amortized $O(\min(\Delta + kk', k^2))$ time [77]), maximal clique enumeration (amortized $O(poly(k))$ time but this algorithm needs exponential space [51]), and bipartite induced subgraph enumeration (amortized $O(k)$ time [78]). The above algorithms achieve amortized constant time enumeration for graphs with bounded degeneracy.

These algorithms use a vertex elimination ordering, called a *degeneracy ordering*. In a degeneracy ordering, every vertex has at most k neighbors that are later in the ordering. Mutula *et al.* [53] has been developed a linear time algorithm to compute the degeneracy and degeneracy ordering. This ordering is a key structure to construct efficient enumeration algorithms In this thesis, we use this ordering in Chapter 5 and Chapter 6.

1.8 Organization

In Chapter 2, we give basic notations used in this thesis. In addition, we introduce basic techniques for the design and analysis of enumeration algorithms. In Chapter 3, we present an constant amortized time enumeration algorithm to enumerate induced matchings in C_4-free graphs. In Chapter 4, we propose an efficient enumeration algorithm for subgraphs and induced subgraphs with bounded girth. In Chapter 5, we give constant amortized time enumeration algorithms for dominating sets in sparse graphs. In Chapter 6, we give an efficient enumeration algorithm for chordal bipartite induced subgraphs. In Chapter 7, we summarize this thesis. In this chapter, we point out the

future direction of this thesis.

Chapter 2

Preliminaries

In this chapter, we introduce basic definitions and notations. In addition, we explain basic techniques to construct and analyze enumeration algorithms.

2.1 Graphs and hypergraphs

Let $G = (V(G), E(G))$ be an *undirected grpah*, or simply a *graph*, where $V(G)$ be a set of *vertices* of G and $E(G) \subseteq V(G) \times V(G)$ is the set of *edges* of G. We denote by n and m the *size* of $V(G)$ and $E(G)$, respectively. We say that u and v are *adjacent* if $E(G)$ includes $\{u, v\}$. A set of adjacent vertices of v is a *open neighborhood* of v. We denote a open neighborhood of v as $N_G(v)$. A set of vertices $N_G(v) \cup \{v\}$ is called *closed neighborhood* and denoted by $N_G[v]$. A pair of vertices u and v are *twin* if $N_G(v)$ is equal to $N_G(u)$. We denote $N_G(v) \cap X$ as $N_X(v)$, where X is a subset of V. A *set of neighbors* of U is defined as $N_G(U) = \bigcup_{u \in U} N_G(u) \setminus U$. Similarly, let $N_G[U]$ be $\bigcup_{u \in U} N_G(u) \cup U$. In addition, an edge $\{u, v\}$ is called an *incident edge* of u. The number of incident edges of v is called the *degree of* v and denoted by $d_G(v)$. The

maximum degree in a graph G is called the *degree of* G and denoted by $\Delta(G)$. For an edge $e = \{u, v\}$, we say that u and v are *end points (or ends)* of e. A set of edges which has v as ends is denoted by $N_G^e(v)$. If there is two edges e and f which has same end points, then we call e and f are *parallel*. An edge $e = \{v, v\}$ is called *self loop*. A graph G is simple if there is no self loops and parallel edges. In this thesis, we assume that G is simple and finite. In what follows, if it is clear from context, we omit the subscript G.

A graph $H = (U, F)$ is a *subgraph of* $G = (V, E)$ if U and F are subset of V and E, respectively. We say that H is a *spanning subgraph* if U is equal to V. A graph $H = (U, F)$ is an *induced subgraph of* $G = (V, E)$ if $F = \{\{u, v\} \in E \mid u, v \in U\}$ and denoted by $H = G[U]$. We say that an induced subgraph $G[U]$ is induced by U. We denote by $G \setminus \{e\} = (V, E \setminus \{e\})$ and $G \setminus \{v\} = G[V \setminus \{v\}]$. For simplicity, we denote by $v \in G$ and $e \in G$ if $v \in V$ and $e \in E$, respectively. A graph is called *k-degenerate* if all induced subgraph has a vertex with the degree at most k [48]. The *degeneracy of a graph* is the minimum value of k. A sequence $P = (v_1, \ldots, v_k)$ of vertices is a *path* if each vertex in P appears at most once and $\{v_i, v_{i+1}\} \in E$ for any $1 \leq i < k$. We also call P an *v_1-v_k path*. The length of P is defined by the number of its vertices minus 1. Then, a sequence $C = (v_1, \ldots, v_k)$ of vertices is a *cycle* if $(v_1, \ldots, v_k)$ is a v_1-v_k path and $\{v_1, v_k\} \in E$. The length of a cycle is defined by the number of its vertices. We say that a graph is *connected* if any pair of u and v has a u-v path. Otherwise, we say that a graph is *disconnected*. For any vertices $u, v \in V$, the *distance between u and v* is defined by the length of a shortest u-v path and denoted by $dist(u, v)$. The distance between edge e and f is defined by the length of a shortest path, i.e., $dist(e, f) = \min\{dist(u, v) \mid u \in e, v \in f\}$. Similarly, the distance between vertex v and edge e is defined by $dist(v, e) = \min\{dist(u, v) \mid u \in e\}$. We define the *neighborhood*

with distance k as $N^k(v) = \{u \in V \mid dist(u,v) = k\}$. In addition, we define the *neighborhood with distance at most* k as $N^{\leq k}(v) = \{u \in V \setminus \{v\} \mid dist(u,v) \leq k\}$. We denote that $N^{\leq k}(v) \cup \{v\}$ as $N^{\leq k}[v]$. $\mathcal{N}(v)$ is the *neighborhood set of the neighbors* defined as $\{N(u) \mid u \in N(v)\}$. A set of edges M is called an *induced matching* if any pair of edges has a distance at least two.

Next, we consider directed graphs. We denote a *directed edge* from a to b as $e = (a,b)$. We call a is a *tail* of e and b is a *head* of e. We call a graph G a *directed graph* when any edge in G has direction. When edge direction is not important, we write $\{a,b\}$ to refer to either the directed edge (a,b) or (b,a). and $N^e(v)$ represents the set of edges having v as either tail or head, which we call *edge neighborhood*. The *directed girth*, or simply *girth*, denoted by $g(G)$ of a graph G, is the length of its smallest cycle. A graph is *acyclic* if it contains no cycle. If G is acyclic, its girth is defined to be ∞; in all other cases, the girth of G is at most $|V(G)|$, i.e., the maximum possible length of a cycle.

Let $\mathcal{H} = (V, \mathcal{E})$ be a *hypergraph*, where V is a set of vertices and $\mathcal{E}$ is a set of subsets of V. We call an element of $\mathcal{E}$ a *hyperedge*. For a vertex v, let $\mathcal{H}(v)$ be the set of edges $\{e \in \mathcal{H} \mid v \in e\}$. A sequence of edges $C = (e_1, \ldots, e_k)$ is a *berge cycle* if there exist k distinct vertices $v_1, \ldots, v_k$ such that $v_k \in e_1 \cap e_k$ and $v_i \in e_i \cap e_{i+1}$ for each $1 \leq i < k$. A berge cycle $C = (e_1, \ldots, e_k)$ is a *pure cycle* if $k \geq 3$ and $e_i \cap e_j \neq \emptyset$ hold for any distinct i and j, where i and j satisfy one of the following three conditions: (I) $|i-j| = 1$, (II) $i = 1$ and $j = k$, or (III) $i = k$ and $j = 1$. A cycle $C = (e_1, \ldots, e_k)$ is a *β-cycle* if the sequence of $(e'_1, \ldots, e'_k)$ is a pure cycle, where $e'_i = e_i \setminus \bigcap_{1 \leq j \leq k} e_j$. We call a hypergraph $\mathcal{H}$ *β-acyclic* if $\mathcal{H}$ has no β-cycles. We call a vertex v a *β-leaf* if $e \subseteq f$ or $e \supseteq f$ hold for any pair of edges $e, f \in \mathcal{H}(v)$. A bipartite graph $\mathcal{I}(\mathcal{H}) = (X, Y, E)$ is a *incidence graph* of a hypergraph $\mathcal{H} = (V, \mathcal{E})$ if $X = V$, $Y = \mathcal{E}$, and E contains an edge $\{v, e\}$ if

$v \in e$, where $v \in V$ and $e \in \mathcal{E}$.

2.2 Complexity of enumeration algorithms

In enumeration algorithm area, there are two measurements to evaluate the efficiency of an algorithm. One is an *input sensitive*. In this measurement, we evaluate the efficiency of algorithm with respect to only an input size. Since the number of solution is typically exponentially larger than an input size, the time complexity of an enumeration algorithm is bounded by exponential time.

The other is an *output sensitive evaluation* [33]. In this measurement, we evaluate the efficiency of an enumeration algorithm with respect to an input size and the number of solutions. In this thesis, we use output sensitive evaluation. An enumeration algorithm is called *amortized polynomial time* if the total time complexity is bounded by $O\left(M \cdot poly(n)\right)$, where M is the number of solution and n is an input size. In addition, an enumeration algorithm is called *polynomial delay* if the maximum interval is bounded by polynomial of an input size. To solve a problem with M solutions, we need M steps since we output M solutions. Hence, an time algorithm which runs in $O\left(M\right)$ time is optimal. We call such algorithm as a *constant amortized time enumeration algorithm*.

2.3 Basic techniques for enumeration algorithm

We introduce several techniques for constructing enumeration algorithms. Binary partition is one of the most famous technique to construct enumeration algorithms. There are many efficient enumeration algorithms applying binary partition [1, 4, 13, 23, 24, 30,

42, 47, 58, 60, 64, 66, 69, 75, 76, 78]. The next famous technique is reverse search. Reverse search have been developed by Avis and Fukuda [3]. The following results are based on reverse search [14, 15, 40, 41, 50, 55, 65, 68, 70, 72, 79]. Finally, we introduce the supergraph technique. In this technique, we construct strongly connected digraph on a solution space. There are many algorithms based on this technique to solve maximum or minimum solution enumeration problems [11, 17, 26, 38, 39, 61].

2.3.1 Binary partition

The basic idea of binary partition is as follows: dividing $\mathcal{S}$ into non empty two sets, $\mathcal{S}_0$ and $\mathcal{S}_1$. If the size of $\mathcal{S}_0$ is one, then output this solution. Otherwise, we divide $\mathcal{S}_0$ recursively. We do the same procedure for $\mathcal{S}_1$. This dividing procedure makes a tree structure $\mathcal{T} = (\mathcal{V}, \mathcal{E})$. We call this tree structure as an *enumeration tree*. Since each internal node of $\mathcal{T}$ divides $\mathcal{S}$ into two sets $\mathcal{S}_0$ and $\mathcal{S}_1$, we can enumerate all solutions.

The bottleneck of binary partition is a division part. It is difficult to divide $\mathcal{S}$ into two non empty set efficiently. In this thesis, our algorithms in Chapter 3 and Chapter 4 are based on binary partition. In these chapters, we show that the size of $\mathcal{S}_1$ and $\mathcal{S}_0$ are sufficiently large if we need much computation time to divide $\mathcal{S}$ into $\mathcal{S}_0$ and $\mathcal{S}_1$. Hence, we can amortized this cost and achieves constant amortized time enumeration.

2.3.2 Reverse search

In reverse search, we define the parent child relation between solutions. Our aim is constructing a spanning tree on a set of solutions. Let $\mathcal{S}$ be a set of solutions and R be a solution which has no parent. If any solution $S \in \mathcal{S} \setminus \{R\}$ become a solution R by applying the parent relation at most k times, then this relationship makes a tree structure $\mathcal{T} = (\mathcal{V}, \mathcal{E})$. We call this tree structure as a *family tree*. In reverse search, we

enumerate all solution by traversing this tree structure $\mathcal{T}$. However, we cannot traverse $\mathcal{T}$ only the parent child relation since we cannot find the children of each node. Hence, we have to define a function to enumerate all children of each node. We summarize the key point of reverse search The definition of the parent child relation and the definition of a function to enumerate all children of each node. If we define the above a relation and a function, then we can traverse $\mathcal{T}$ and enumerates all solutions. In this thesis, our algorithms in Chapter 5 and Chapter 6 are based on reverse search.

2.3.3 The supergraph technique and proximity search

In the supergraph technique, we define the neighbor of each solution. Hence, a set of solution and this neighborhood relation make a directed graph. By traversing this directed graph, we enumerate all solutions.

Cohen *et al.* have been developed a general framework to enumerate all maximal subgraphs for hereditary and connected hereditary graph properties. This framework is based on the supergraph technique and guarantees that a constructed supergraph is strongly connedted. In this framework, we have to slove some enumeration problem. This subproblem is called an *input restricted problem.* If we can solve an input restricted problem in output polynomial time, then we can enumerate maximal subgraphs in output polynomial time. Unfortunately, it is difficult to solve an input restricted problem in polynomial time since an input restricted problem is an enumeration problem.

To overcome this difficulty, Conte and Uno have been developed a new technique to construct a sparse strongly connected supergraph. This technique is called *proximity search* [17]. In proximity search, the out degree can be reduced to polynomial size if a solution has good ordering. Hence, in proximity search, we compute all neighborhood in polynomial time for each vertex on a supergraph.

2.4 The time complexity analysis techniques

Enumeration algorithms based on reverse search or binary partition make a recursion tree. Suppose that an algorithm outputs at least one solution in each node of this recursion tree. The amortized time complexity of such an enumeration algorithm is $O(T(X))$ time, where X is a node in a recursion tree and $T(X)$ is the time complexity of node X. We introduce the technique to reduce the amortized complexity. In what follows, suppose that an enumeration algorithm makes a recursion tree.

2.4.1 Amortization by children and grandchildren

To reduce the amortized time complexity, we consider the number of children and grandchildren. Let $\mathcal{A}$ be an enumeration algorithm. Let X be a node in a recursion tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$. the number of children and grandchildren of X are denoted by $ch(X)$ and $gch(X)$, respectively. In each node X, $\mathcal{A}$ demands $T(X) = O(ch(X) + gch(X))$ time. In this case, the total time complexity of this algorithm is bounded by $O\left(\sum_{X \in \mathcal{T}} ch(X) + gch(X)\right)$. Since the sum of the number children and grandchildren is bounded by $O(|\mathcal{V}|)$, the amortized time complexity of this algorithm is $O(1)$ if the size of $\mathcal{V}$ is bounded by $O(M)$, where M is the number of solutions. This analysis is simple. However, we use this analysis many times in this thesis.

2.4.2 Push out amortization

Uno has been developed the analysis technique, called *push out amortization* [73]. In push out amortization, the key point is the total computation time of children nodes. Uno shows that if any internal node satisfies the following condition, called *push out condition (PO condition)*, then, the amortized time complexity of this algorithm is

$O(T^*)$, where T^* is the time complexity of a leaf node. Let X be a node in an enumeration tree $\mathcal{T}$. PO condition is as follows: $T(ch(X)) \geq \alpha T(X) - \beta T^* |ch(X) + 1|$, where α is a real number greater than 1, β is a real number greater than or equal to 0, $T(X)$ is the computational time in node X, and $T(ch(X))$ is the total computation time of children nodes. In this analysis, we distribute the computational cost recursively. If PO condition is satisfied in any node in $\mathcal{T}$, then total computation time of this algorithm is dominated by the bottom part of $\mathcal{T}$. Since each leaf has $O(T^*)$ cost, the amortized time complexity is $O(T^*)$.

Chapter 3

Enumeration of Induced Matchings

3.1 Introduction

In this chapter, we propose an efficient enumeration algorithm for induced matchings. An *induced matching* is a set of edges such that any pair of edges has no adjacent endpoints. We show that our algorithm enumerates all solutions in constant amortized time if graphs have no cycles with length four. The maximum induced matching problem is a famous NP-hard problem [63]. Moreover, Moser and Sikdar show that the maximum induced matching problem is fixed parameter tractable for graphs with girth six or more [54]. We evaluate our algorithm by *output sensitive manner* [33]. In other words, we evaluate the time complexity with respect to the size of input and the

---

This chapter is based on "Efficient Enumeration of Induced Matchings in a Graph without Cycles with Length Four," [45] by the same authors, which appeared in the Proceedings of IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, Copyright(c) 2017 IEICE. The material in this chapter was presented in part at the Proceedings of IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences [45], and all the figures of this chapter are reused form [45] under the permission of the IEICE.

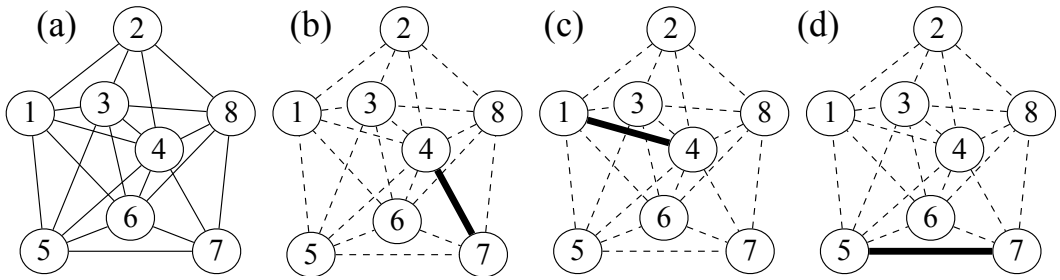


Figure 3.1: An example of an intractable input graph.

number of solutions. In this measurement, we evaluate the efficiency of an enumeration algorithm with respect to an input size and the number of solutions.

Our algorithm is binary partition based algorithm. Hence, we spend considerable amount time in generating all child problems if the graph is dense, In Figure 3.1, (a) is an input graph. (b), (c), and (d) are induced matchings in this graph. Thick solid lines are included an induced matching, dotted lines are not included. In this case, it is difficult to enumerate all induced matching efficiently since the number of child problems is 0 even if we select any edge. We need much time to check it. In such a case, it is difficult to efficiently enumerate all solutions because the number of child problems is small. The number of child problems is important for efficient enumeration. If it is adequate, considerable amount of time is available for generating child problems. Thus, efficient enumeration of dense graphs is difficult, and we therefore consider not dense graphs. We define the induced matching enumeration problem as follows.

Problem 1 (The induced matching enumeration problem). *Output all induced matching in a given graph G without duplicates.*

Main result

We present an efficient induced matching enumeration algorithm `EIM` for C_4-free graphs. This algorithm based on simple binary partition. The key point is the number of children of siblings. In each iteration, we need more than $O(1)$ time. However, this computational time is corresponding to the number of children of siblings if an input graph is C_4-free.

3.2 A basic algorithm based on binary partition

A *binary partition method* is a method to construct an enumeration algorithm which enumerates all solutions by dividing a search space into two disjoint search spaces recursively. Let $\mathcal{A}$ be an enumeration algorithm based on binary partition. We call a dividing step an *iteration*. Let G, $\mathcal{M}(G)$, and X be an input graph, the set of solutions for G, and an iteration of the algorithm, respectively. Let $\mathcal{S}(X)$ be the set of solutions included in a search space of X. In the initial state, $\mathcal{S}(X) = \mathcal{M}(G)$ holds. At the initial iteration, the algorithm selects an edge e in G such that e satisfies $|\mathcal{S}_0(X)| \geq 1$ and $|\mathcal{S}_1(X)| \geq 1$ where $\mathcal{S}_0(X) = \{M \in \mathcal{S}(X) \mid e \notin M\}$ and $\mathcal{S}_1(X) = \{M \in \mathcal{S}(X) \mid e \in M\}$. Note that $\mathcal{S}(X) = \mathcal{S}_0(X) \cup \mathcal{S}_1(X)$ holds. An algorithm $\mathcal{A}$ recursively applies this procedure until all edges are selected.

Next, we introduce a *binary enumeration tree* $\mathcal{T} = (\mathcal{V}, \mathcal{E})$. Here, $\mathcal{V}$ is the set of iterations of $\mathcal{A}$ and $\mathcal{E}$ is a subset of $\mathcal{V} \times \mathcal{V}$. For any iteration X, we define the edge set E_X as follows: $E_X = \bigcap_{M \in \mathcal{S}(X)} M$. In other words, E_X is the set of edges included by all solutions in $\mathcal{S}(X)$. For any iterations X and Y, Y is a *child* of X if $E_Y \subset E_X$ and $|E_X \setminus E_Y| = 1$ hold. We call X is the *parent* of Y. For any iteration X, we define iterations $X.1$ and $X.0$ with the set of solutions $\mathcal{S}_1(X)$ and $\mathcal{S}_0(X)$, respectively. That

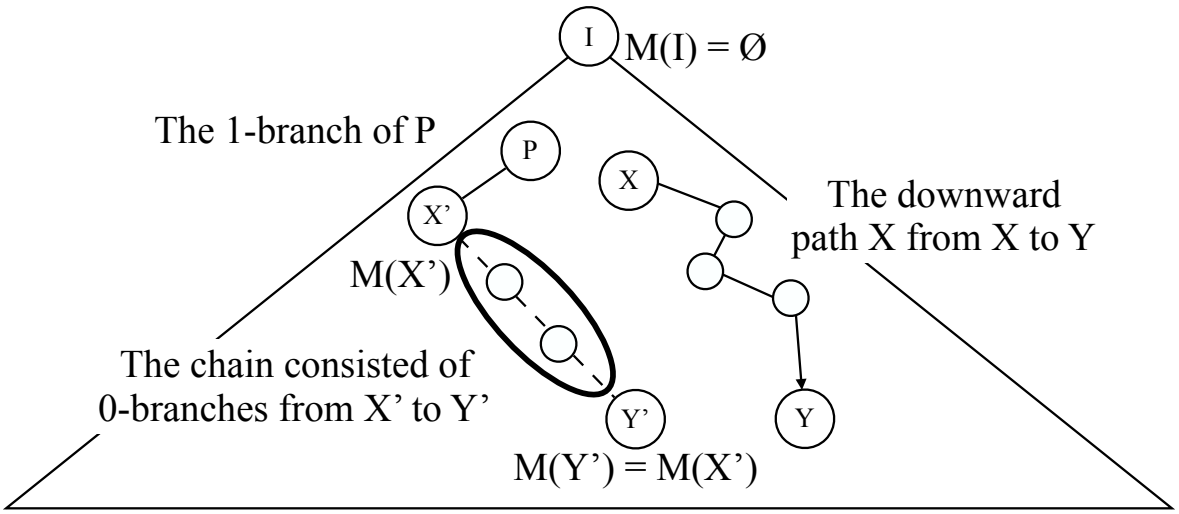


Figure 3.2: The downward path X and the chain consisted of 0-branches in $\mathcal{T}$.

is, $X.1$ and $X.0$ are the children of X. In particular, $X.1$ is the 1-*child of* X, and $X.0$ is the 0-*child of* X. In addition, we call edges $e = \{X, X.0\}$ and $f = \{X, X.1\}$ in $\mathcal{E}$ a 0-*branch* and a 1-*branch*, respectively. In the binary enumeration tree $\mathcal{T}$, we call an iteration with children an *internal iteration*, and an iteration without children a *leaf iteration*. Moreover, an iteration X is the *root iteration* if there is no iteration that has X as a child, that is, X is the first iteration called by $\mathcal{A}$. For the simplicity, we call the binary enumeration tree the *enumeration tree*.

For any iterations X and Y, a *downward path* from X to Y in $\mathcal{T}$ is a sequence of iterations $\mathcal{L} = (X = X_0, \ldots, X_k = Y)$, where for each $i = 1, \ldots, k$, X_k is a child of X_{k-1}. For any iterations X and Y, if there is a downward path $\mathcal{L}$ from X to Y, then X is an *ancestor* of Y and Y is a *descendant* of X. $X \preceq Y$ if X is an ancestor of Y. For any iteration Y, the set $\{X \mid X \preceq Y\}$ is a *chain* of Y. When iterations X and Y belong to a same chain, X and Y are *comparable*. In a chain $\mathcal{L}$, we call an iteration X

Algorithm 1: An algorithm enumerating all induced matchings in C_4-free graphs in constant amortized time.

```
Procedure EIM (G = (V, E))
    RecEIM (∅, G) ;
Procedure RecEIM (M, G)
    if E(G) = ∅ then
        Output M;
        return;
    The vertex v has the maximum degree in G;
    RecEIM (M, G \ {v});                                  //0-child
    G' ← G \ N[v];
    for e ∈ D^0(v) do
        RecEIM (M ∪ {e}, G' \ Sect^2(e, v));              //i-child
    Restore edges in D^0(v) ∪ D^1(v);
    return;
```

is the *minimum element* and the *maximum element* if X is the head of $\mathcal{L}$ and is the tail of $\mathcal{L}$, respectively. In Figure 3.2, we show an example of a downward path and a chain in $\mathcal{T}$. The solid line from X to Y represents a downward path. Consequently, $X \preceq Y$ holds. The dotted line represents a chain consisting of 0-branches. X' is a top of the chain since X' is a 1-child of P. Y' is a bottom of the chain since Y' does not have a child.

3.3 The algorithm for C_4-free graphs

We show the algorithm `EIM` in *Algorithm* 1. `RecEIM` selects the vertex v with the maximum degree. We call such a vertex v a *pivot* on X. For any iteration X of `RecEIM`, let $M(X)$, $S(X)$ and $G(X)$ be the current induced matching as solution, the set of vertices that are selected in the ancestor iterations of X as the pivots, and a graph $G(X) = G[V \setminus (N(V(M(X))) \cup S(X))]$, respectively. An edge e in G is a *conflict edge* if there exists an edge $f \in M(X)$ satisfying $dist(e, f) \leq 1$. Otherwise, we say that e is a *safe edge*. That is, $G(X)$ is a graph removed all conflict edges with $M(X)$ and $S(X)$ from G. Let $k \in \{0, 1, 2\}$ and $\ell \in \{0, 1, 2\}$. We define the concentric structure $D^{k,\ell}(v)$ as follows:

$$\begin{aligned} D^{k,\ell}(v) = \{\{x, y\} \in E(G(X)) \mid dist_{G(X)}(x, v) &= k, \\ dist_{G(X)}(y, v) &= \ell\}. \end{aligned}$$

$D^k(v)$ denotes $D^{k,k}(v) \cup D^{k,k+1}(v)$. We call an edge $e \in D^{k,\ell}(v)$ a *k-ℓ edge of v* and an edge $e \in D^k(v)$ a *k-$*$ edge of v*. Figure 3.3 (a) shows an example of 0-1 edges, 1-$*$ edges, and 2-$*$ edges. The distance between two vertices is defined by the length of shortest path in not G but $G(X)$. For any 0-1 edge $e_i = \{v, u_i\}$ of a pivot v, we define $Sect^k(e_i, v)$ as follows:

$$\begin{aligned} Sect^k(e_i, v) = \{f \in D^k(v) \mid dist_{G(X)}(e_i, f) &= k - 1, \\ dist_{G(X)}(v, f) &= k\}. \end{aligned}$$

We show an example of $Sect^k(e_i, v)$ in Figure 3.3. In case (a), the shaded area indicates the area of edges to be removed when calling type-0 child. In case (b), the shaded area indicates the area of edges to be removed when calling type-1 child. The area

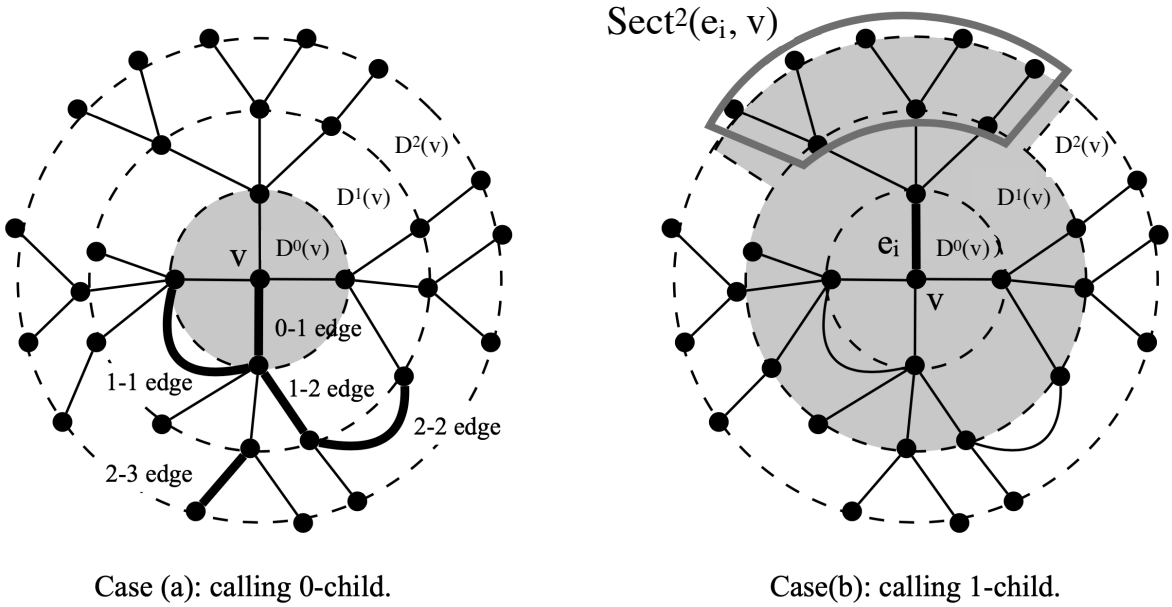


Figure 3.3: An example of dividing edges by the pivot v in the graph G. In (b), the area surrounded by the solid line represents $Sect^2(e_i, v)$.

surrounded by the solid line represents $Sect^2(e_i, v)$. Note that, from the assumption, G has no self loops, thus, $D^{0,0}(v) = \emptyset$.

`RecEIM` outputs $M(X)$ as a solution if no edge can be added to $M(X)$ from $G(X)$ and quits X. `RecEIM` skips this step and execute the following steps if there is an edge that can be added to $M(X)$. Next, `RecEIM` divides a solution set $\mathcal{S}(X)$ into $d(v) + 1$ disjoint sets $\mathcal{S}_0(X), \ldots, \mathcal{S}_{|d(v)|}(X)$. Let e_i be the ith edge incident to v for $i \in \{1, \ldots, d_{G(X)}(v)\}$. $\mathcal{S}_i(X) \subseteq \mathcal{S}(X)$ is the set of solutions including e_i, and $\mathcal{S}_0(X) = \mathcal{S}(X) \setminus \bigcup_{i=1,\ldots,d_{G(X)}(v)} \mathcal{S}_i(X)$ is the set of solutions not including edges adjacent to v. $X.i$ denotes the ith child iteration of X that receives $M(M(X)) \cup \{e_i\}$. Also, we call $X.0$ *type*-0 *child* and $X.i$ *type*-1 *child* for $i \neq 0$. We call the branch from X to the

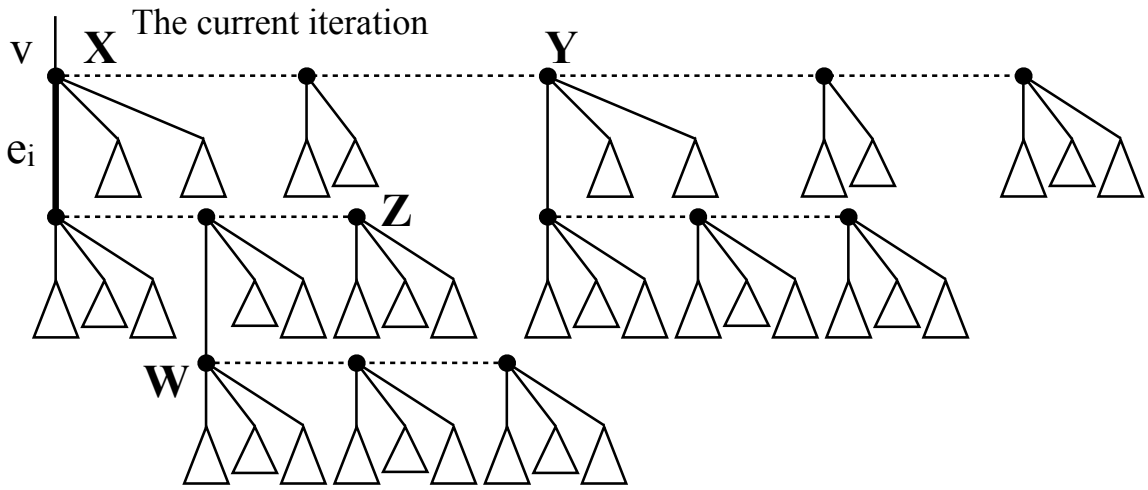


Figure 3.4: An example of an enumeration tree made by `EIM`.

type-0 child the 0-*branch* and a branch from X to type-1 child a 1-*branch*. Let $\mathcal{T}$ be an enumeration tree made by `EIM`. Note that $\mathcal{T}$ is a multi-way tree. In Figure 3.4, we show an example of $\mathcal{T}$. Black dots represent vertices in the enumeration tree, in other words, iterations. Triangles represent subtrees. Dotted lines and solid lines represent 0-branches and 1-branches. A label v and e_i show a pivot and a selected edge in $G(X)$.

Lemma 1. *Let X and Y be any iterations. If $X \preceq Y$ then $M(X) \subseteq M(Y)$ and $E(G(X)) \supseteq E(G(Y))$ hold.*

Proof. There is a downward path $\mathcal{L}$ from X to Y since $X \preceq Y$. It is obvious that `EIM` does not remove any edge in $M(X)$ from $M(X')$ in $X' \in \mathcal{L}$. On the other hands, `EIM` does not add any edge to $E(G(X))$ to $E(G(X'))$ in $X' \in \mathcal{L}$. $\square$

3.4 Correctness of the algorithm

We show the correctness of Algorithm 1. The next lemma implies that $M(X) \cup \{e\}$ is an induced matching in G for any edge $e \in D^0(v)$.

Lemma 2. *Let G be an input graph, X be any iteration in the algorithm* `EIM` *in Algorithm 1, and e be any edge in $D^0(v)$. Then, $M(X) \cup \{e\}$ is an induced matching in G.*

Proof. We prove by contradiction. Assume that there is a conflict edge $f \in M(X)$ with e. From Lemma 1, there is an iteration $Y \preceq X$ such that f is added to $M(Y)$. Since f conflict with e, either (1) e is adjacent to f or (2) e is not adjacent to f and there is an edge g in $G(Y)$ that is adjacent to both e and f. We consider case (1). By the definition of $G(Y)$, if `EIM` adds f to $M(Y)$ in Y, edges adjacent to f are not in $G(W)$, for any descendant iteration W of Y. This contradicts the assumption. Next, we consider case (2). If $g \in G(Y)$ holds, then $dist_{G(Y)}(e, g) = 1$. This implies that for any descendant iteration W of Y, $e \notin G(W)$. On the other hands, if $g \notin G(Y)$, then $G(Y)$ is not induced subgraph since $e, f \in G(Y)$ and $g \notin G(Y)$. This also contradicts the assumption. Hence the statement holds. □

Since `EIM` outputs a solution in a leaf iteration and $M(X)$ is induced matching for each iteration $X \in \mathcal{T}$, the following corollary holds from Lemma 2.

Corollary 3. *The algorithm* `EIM` *in Algorithm 1 outputs only induced matchings.*

Next, we consider a method for obtaining $G(X.i)$.

Lemma 4. *Let X be an iteration in the algorithm* `EIM` *in Algorithm 1. Then, $G(X.0) = G(X) \setminus \{v\}$ holds.*

Proof. Since $M(X) = M(X.0)$, there is no edge in $G(X.0)$ such that one of its endpoint is v. Thus, the statement holds. □

Lemma 5. *Let X be any iteration in the algorithm* `EIM` *in Algorithm 1 and i be positive integer. Then, $G(X.i) = G(X) \setminus (D^0(v) \cup D^1(v) \cup Sect^2(e_i, v))$ holds.*

Proof. For any edge f in G, we show that the following cases are equivalent: (1) f is conflict edge of e and (2) f belongs to $D^0(v) \cup D^1(v) \cup Sect^2(e_i, v)$. Let $e_i = \{v, v_i\}$ and $f = \{u, w\}$. Without loss of generality, we can assume that $dist_G(u, v) \leq dist_G(w, v)$. If $f \notin D^0(v) \cup D^1(v) \cup Sect^2(e_i, v)$, then $1 < dist_G(u, v) \leq dist_G(w, v)$ and $1 < dist_G(u, v_1) \leq dist_G(w, v_1)$ hold. Hence, f does not conflict with e_i. On the other hands, if $f \notin D^0(v) \cup D^1(v) \cup Sect^2(e_i, v)$ then f obviously conflicts with e_i. Hence, the statement holds. □

Lemma 4 and Lemma 5 imply that `EIM` correctly computes $G(X.i)$ in X.

Lemma 6. *The algorithm* `EIM` *in Algorithm 1 outputs solutions without duplication.*

Proof. Let X and Y be two distinct leaf iterations. We proceed by contradiction. Suppose that $M(X) = M(Y)$. From the assumption, X and Y are incomparable. Hence, without loss of generality the lowest common ancestor of X and Y always exists. Let Z be the lowest common ancestor of X and Y. We consider the following two cases. (1) Suppose that both X and Y are descendants of the type-0 child $Z.0$ of Z. This contradicts that Z is the lowest common ancestor of X and Y. (2) Suppose that at least one of X and Y is a descendant of a type-1 child of Z. Without loss of generality, X is a descendant of the ith child $Z.i$ of Z. If W is $Z.j$ or is any descendant of $Z.j$ where $j \neq i$, then $M(W)$ does not include e_i and this contradicts $M(X) = M(Y)$. Hence, the statement holds. □

Lemma 7. *The algorithm* `EIM` *in Algorithm 1 outputs all solutions in* $\mathcal{M}$.

Proof. Let $\mathcal{T}$ be an enumeration tree, X be any iteration on $\mathcal{T}$, and v be the pivot on X. From Lemma 4 and Lemma 5, `EIM` correctly divides a solution set $\mathcal{S}(X)$ in X into $\mathcal{S}_0(X), \ldots, \mathcal{S}_{d(v)}(X)$. If $|\mathcal{S}(X)| = 1$, then `EIM` outputs $\mathcal{S}(X)$. The statement holds. □

From corollary 3, Lemma 6, and, Lemma 7, the next theorem holds.

Theorem 8. *The algorithm* `EIM` *in Algorithm 1 enumerates all solutions without duplication.*

3.5 Amortized analysis of the time complexity

In this section, we show that `EIM` enumerates all induced matchings in constant time per solution for C_4-free graphs, where C_4-free graphs are graphs which has no cycle with length four as a *subgraph*, not an induced subgraph. If the degree of the pivot on X is less than three, then `EIM` obviously runs in the constant amortized time per solution. Thus, we assume the degree of pivot is at least three. It is redundant to process safe and conflict edges independently among siblings of the same parent. To avoid this, we simultaneously process these edges by using concentric structures $D^{k,\ell}(v)$ around the pivot on X.

We consider the data structure $\mathcal{L}ist(G(X))$ to efficiently extract the set of vertices whose degree is k when we are given k. We use $\mathcal{L}ist(G(X))$ to find the vertex v with the maximum degree. We define $\mathcal{L}ist(G(X))$ as follows: $\mathcal{L}ist(G(X)) = \{L_0, \ldots, L_{\Delta(G(X))}\}$, where for any $0 \leq i \leq \Delta(G(X))$, $L_i = \{v \in V(G) \mid d_{G(X)}(v) = i\}$. The lists in $\mathcal{L}ist(G(X))$ are implemented by doubly-linked lists. For brevity, we write $x \in \mathcal{L}ist(G(X))$ if there is L_i such that $x \in L_i$ in $\mathcal{L}ist(G(X))$. Usually, when we search

a position of a vertex v in $\mathcal{L}ist(G(X))$, we need linear time for the size of $\mathcal{L}ist(G(X))$ since L_i is implemented by doubly-linked list. Now, we prepare an array $P_{G(X)}$ to find a position of v in $\mathcal{L}ist(G(X))$ in constant time. $P_{G(X)}$ store address of each vertex in $\mathcal{L}ist(G(X))$. We can find position of any vertex in $\mathcal{L}ist(G(X))$ in constant time. For any input graph G, we can compute $\mathcal{L}ist(G)$ in $O(|V(G)|)$ time by using bucket sort. We can implement $\mathcal{L}ist(G(X))$ such that extracting any vertex from $\mathcal{L}ist(G(X))$ can be done in constant time.

Lemma 9. *Let X be any iteration in $\mathcal{T}$. Then, we can find the pivot in constant time by using $\mathcal{L}ist(G(X))$.*

Proof. It is obvious that $\bigcup_{L_i \in \mathcal{L}ist(G(X))} = V(G)$ holds. $i_* = \underset{i \in \{0,\ldots,\Delta(G(X))\}}{\arg\max} (L_i \neq \emptyset)$, that is, L_{i_*} is the non empty list with the maximum index in $\mathcal{L}ist(G(X))$. Now, the pivot on X is in L_{i_*} since $\Delta(G(X)) = i_*$. Hence, `EIM` can obtain v in constant time by extracting a vertex from the tail of L_{i_*}. □

Lemma 9 shows that we can find v in constant time. Next, we consider management of the data structure $\mathcal{L}ist(G(X))$. When we delete v in $G(X)$, we update $\mathcal{L}ist(G(X))$ to $\mathcal{L}ist(G(X) \setminus \{v\})$. It is completed in $O\left(d_{G(X)}(v)\right)$ time using $P_{G(X)}$. Thus, we manage $\mathcal{L}ist(G(X))$ in $O\left(d_{G(X)}(v)\right)$ time to use $P_{G(X)}$ and find a vertex v with the maximum degree. Next, we define the edge set $D^{\leq 2}(v)$ as follows: $D^{\leq 2}(v) = \bigcup_{i=0,1,2} D^i(v)$, i.e. $D^{\leq 2}(v)$ consist of all edges with distance less than two from v.

Lemma 10. *Let G be a C_4-free graph, v be the pivot on an iteration X, and, u be a vertex satisfying $dist_G(u, v) = 2$. Then, the number of edges whose endpoint is u in the set of 1-2 edges of v is exactly one.*

Proof. We prove by contradiction. Let $f_1 = \{u, w_1\}$ and $f_2 = \{u, w_2\}$ be two distinct 1-2 edges whose end point is u. We assume $w_1 \neq w_2$. Since f_1 and f_2 are 1-2 edges

and $dist_G(u, v) = 2$, $dist_G(v, w_1) = dist_G(v, w_2) = 1$ holds. Thus, there exist two edges $e_1 = \{v, w_1\}$ and $e_2 = \{v, w_2\}$. Hence, there is a cycle $(v, w_1, u, w_2,)$ in G. This contradicts that G is a C_4-free graph. Hence, the statement holds. □

Lemma 11. *Let G be a C_4-free graph and v be the pivot on an iteration. Then, the following inequality holds:* $\sum_{e \in D^0(v)} |Sect^2(e, v)| \leq 2\,|D^2(v)|$.

Proof. We show that $\bigcup_{e \in D^0(v)} Sect^2(e, v) = D^2(v)$. Let $f = \{x, y\}$ be an edge in $Sect^2(e, v)$. By definition, $f \in D^2(v)$ holds. Without loss of generality, we can assume that $dist_G(x, v) = 2$. Since $dist_G(x, v) = 2$, there is a vertex w satisfying $dist_G(x, w) = dist_G(w, v) = 1$. By definition, f belongs to $Sect^2(\{w, v\}, v)$. Hence, $\bigcup_{e \in D^0(v)} Sect^2(e, v) = D^2(v)$ holds.

Next, we assume that for any 2-$*$ edge $f \in D^2(v)$, f belongs to the following three sets; $Sect^2(e_1, v)$, $Sect^2(e_2, v)$, and $Sect^2(e_3, v)$, where $e_1, e_2, e_3 \in D^0(v)$. Then, by the definition of $Sect^2(e_i, v)$, $dist_G(e_i, f) = 1$ holds for $i \in \{1, 2, 3\}$. Hence, there is some $g_i = \{x_i, y_i\}$ satisfying $x_i \in e_i$ and $y_i \in f$. By the definition of e_i, $dist_G(v, x_i) = 1$ and $dist_G(v, y_i) = 2$ hold. This implies that g_i is a 1-2 edge that shares the end point with f. By the pigeonhole principle, one of the end points of f has at least two 1-2 edges. This contradicts with Lemma 10, hence the statement holds. □

In the remaining of this section, we show that `EIM` enumerates all solutions in constant amortized time per solution. To show the complexity, we show that the ratio between the number of 1-child iterations and 0-child iterations is constant. Let X be any iteration in $\mathcal{T}$ and v be the pivot on X. Suppose that $e = \{x, y\}$ is any edge in $D^{\leq 2}(v)$ and $f = \{v, z\}$, where v, x, y, and z are mutually distinct. We denote by $C(X, e)$ a descendant iteration of X such that $Y = C(X, e)$ is the top of the chain including X and receives $M(Y)$ defined as follows.

(1) If e is a 0-1 edge, then $M(Y) = M(X) \cup \{e\}$.

(2) If e is a 1-1 edge, then $M(Y) = M(X) \cup \{f\}$.

(3.a) If e is a 1-2 edge and $Sect^2(f, v) = \emptyset$, then $M(Y) = M(X) \cup \{e\}$.

(3.b) If e is a 1-2 edge and $Sect^2(f, v) \neq \emptyset$, then $M(Y) = M(X) \cup \{f', g\}$, where $f' \in D^0(v)$ and $g \in Sect^2(f, v)$ such that $f' \neq f$ and $dist_G(f', g) = 2$.

(4) If e is 2-$*$ edge, then $M(Y) = M(X) \cup \{e, h\}$, where h is an edge such that h is adjacent to v and $dist_G(e, h) = 2$.

We call $C(X, e)$ the *corresponding iteration to X w.r.t e*. The next lemma shows that $C(X, e)$ satisfying the above conditions always exists.

Lemma 12. *For any iteration X and $e \in D_v^{\leq}(2)$, there always exists the corresponding iteration $C(X, e)$ to X w.r.t e.*

Proof. Let $e = (x, y)$. From Lemma 7, to proof the lemma, all we have to do is show that $M(C(X, e))$ is a solution. If (1) or (3.a) holds, then $M(C(X, e))$ is obviously an induced matching since $e \in D^{\leq 2}(v)$. Next, we consider condition (2). There are two edges $\{v, x\} = f$ and $\{y, v\}$ since e is a 1-1 edge. Since $f \in D^{\leq 2}(v)$, $M(C(X, e))$ is a solution. Next, we consider condition (3.b). Let $f = \{v, x\}$. Since e is a 1-2 edge, such edge f always exists. Let g be any edge in $Sect^2(f, v)$. From Lemma 10, at least one of the end points of g connects exactly one 1-2 edge. Hence, there is an edge $f' \neq f$ that is adjacent to v and satisfies $dist_G(f', g) = 2$ since the degree of v is at least three. Moreover, $\{f', g\}$ is an induced matching since $dist_G(f', g) = 2$. Hence, $M(C(X, e))$ is an induced matching. Finally, we consider condition (4). Since $d_G(v) \geq 3$, there exists an edge h that is adjacent to v and satisfies $dist_G(e, h)$. Hence, $M(C(X, e))$ is an induced matching. $\square$

In the following lemmas, for any iteration X, we show the number of pairs of an iteration and an edge whose corresponding iteration is X is constant.

Lemma 13. *If a graph G is C_4-free, then each* 0-1 *edge $\{u, v\}$ of v is adjacent to at most one* 1-1 *edge.*

Proof. We show the lemma by contradiction. Suppose there are two distinct 1-1 edges $f = \{u, w\}$ and $g = \{u, x\}$ that are adjacent to a 0-1 edge e. By the definition of a 1-1 edge, $\{u, w, x\} \subseteq N(v)$. Hence, there exist two distinct edges $f' = \{v, w\}$ and $g' = \{v, x\}$. However, this implies that there exist a 4-cycle (v, w, u, x). This contradicts that G is C_4-free. Hence, the statement holds. □

Lemma 14. *Let X and e be a pair of an iteration on $\mathcal{T}$ and an edge in $D^{\leq 2}(v)$ satisfying condition (3.a), and Y be any iteration on $\mathcal{L}$ from X to $C(X, e)$ such that Y satisfies $C(Y, e) = C(X, e)$. Then, the number of such Y is at most two.*

Proof. We first show that $\mathcal{L}$ consists of 0-branches except the end of $\mathcal{L}$. By the definition of $C(X, e)$, the end of $\mathcal{L}$ is a 1-branch since $C(X, e)$ is the top of a chain. Next, $\mathcal{L}$ includes exactly one 1-branch since $M(C(X, e)) = M(X) \cup \{e\}$. Hence, $\mathcal{L}$ consists of only 0-branch other than the end of $\mathcal{L}$.

By using the above observation, we prove by contradiction. Suppose that there exists three distinct iterations Y_1, Y_2, and Y_3 such that they satisfy condition (3.a) and $C(X, e) = C(Y_1, e) = C(Y_2, e) = C(Y_3, e)$. Thus, for any $i = 1, 2, 3$, e is a 1-2 edge of v_i that is the pivot of Y_i. Let $e = \{x, y\}$ and f_i be an edge such that f_i shares the end point with e and f_i is adjacent to v_i. Without loss of generality, we can assume $f_1 = \{x, v_1\}$ and $Y_1 \preceq Y_2$, $Y_1 \preceq Y_3$. For $j = 2, 3$, if $G(Y_j)$ has the edge $f_j = \{y, v_j\}$, then this contradict with $Sect^2(f_i, v) = \emptyset$. Hence, v_j is a neighbor of x. In Y_1, the number of 1-1 edges adjacent to f_1 is at most one from Lemma 13. Hence, at least one

of f_2 and f_3 is a 1-2 edge. Without loss of generality, we can assume that f_2 is a 1-2 edge. Since $Sect^2(f_1, v) = \emptyset$, $D_{G(Y_1)}(v_3) = 1$. In addition, in Y_3, $d_{G(Y_3)}(x) \geq 2$ since x is adjacent to y and v_3. However, this contradicts with $d_{G(Y_3)}(v_3) \geq d_{G(Y_3)}(x)$. Hence, the statement holds. □

Lemma 15. *Let X be any iteration in $\mathcal{T}$. Then, the number of pairs an iteration Y and an edge e satisfying $C(Y, e) = X$ is at most six.*

Proof. Let $\mathcal{L}$ be the path from the root iteration I on $\mathcal{T}$ to X and X' be the parent iteration of X. Without loss of generality, we can assume that X is the top of a chain by the definition of $C(Y, e)$. Let X'' be the parent of the top of a chain including X' and $\mathcal{L}'$ be the path from X'' to X'. By the definition of $C(Y, e)$, Y satisfying $C(Y, e) = X$ exists only on $\mathcal{L}'$. If Y is X', then the number of edges e' satisfying $C(Y, e') = X$ is at most two by conditions (1) and (2). If Y is X'', then the number of edges e' satisfying $C(Y, e') = X$ is at most two by conditions(3.b) and (4). If Y is not X' and X'', then the number of Y satisfying $C(Y, e) = X$ is at most two from Lemma 14. Hence, the statement holds. □

From Lemma 13, Lemma 14, and Lemma 15, for any iteration $X \in \mathcal{T}$, the number of pairs of an iteration Y and an edge e such that $C(Y, e) = X$ is constant. Next, the following lemmas show that total computation time in `EIM` is $O(|\mathcal{T}|)$ time. Let $F(X)$ be $\bigcap_{i=0,\ldots,\Delta(G(X))} E(G(X.i))$. That is, $F(X)$ is the set of edges that are shared by all child iterations of X.

Lemma 16. *Let v be the pivot on an iteration X in $\mathcal{T}$. Then, $E(G(X)) \setminus F(X) = D^{\leq 2}(v)$.*

Proof. Let $\{e_1, \ldots, e_{d_{G(X)}(v)}\}$ be the set of edges that are adjacent to v. We show $F(X) = E(G(X)) \setminus D^{\leq 2}(v)$. Firstly, we show $F(X) \subseteq E(G(X)) \setminus D^{\leq 2}(v)$. For any $i =$

$0, \ldots, d_{G(X)}(v)$, $M(G(X.i))$ includes $e_i = \{v, u_i\}$ by definition. Hence, $F(X)$ does not include conflict edges of e_i. In addition, each edge $f \in F(X)$ satisfies $dist_{G(X)}(f, v) \geq 2$ and $dist_{G(X)}(f, u_i) \geq 2$. Therefore, f is not included in $D^{\leq 2}(v)$. Secondly, we show $F(X) \supseteq E(G(X)) \setminus D^{\leq 2}(v)$. Let g be any edge in $E(G(X)) \setminus D^{\leq 2}(v)$. By definition, $dist_{G(X)}(g, e_i) \geq 2$ holds for any e_i. Therefore, $g \in F(X)$ since $g \in E(G(X.i))$. Now, $D^{\leq 2}(v) \subseteq E(G(X))$ and $F(X) \cap D^{\leq 2}(v) = \emptyset$. Hence, the statement holds. □

Lemma 17. $\sum_{X \in V(\mathcal{T})} |E(G(X)) \setminus F(X)|$ *is bounded by* $O(|\mathcal{T}|)$.

Proof. Let X be any iteration and v be the pivot on X. The number of iterations $Y = C(X, e)$ is at most $\left|D^{\leq 2}(v)\right|$, where e is an edge in $D^{\leq 2}(v)$. Hence, the number of all corresponding iterations is equal to $\sum_{X \in V(\mathcal{T})} |E(G(X)) \setminus F(X)|$ since $E(G(X)) \setminus F(X) = D^{\leq 2}(v)$ from Lemma 16 together with that a pair of an internal iteration X and an edge $e \in D^{\leq 2}(v)$ corresponds to exactly one iteration $C(X, e)$. Next, we consider the number of all corresponding iterations. The number of pairs of an iteration Y and an edge e such that $C(Y, e) = X$ is at most constant from Lemma 15. Since the number of internal iteration is $|\mathcal{T}|$, the number of all corresponding iterations is bounded by $O(|\mathcal{T}|)$, Hence, the statement holds. □

From Theorem 8, Lemma 10, and Lemma 17, we show the following theorem.

Theorem 18. *The algorithm* `EIM` *in Algorithm 1 enumerates all induced matchings in constant amortized time per solution in a* C_4*-free graph* G *after* $O(|V| + |E|)$ *preprocessing time.*

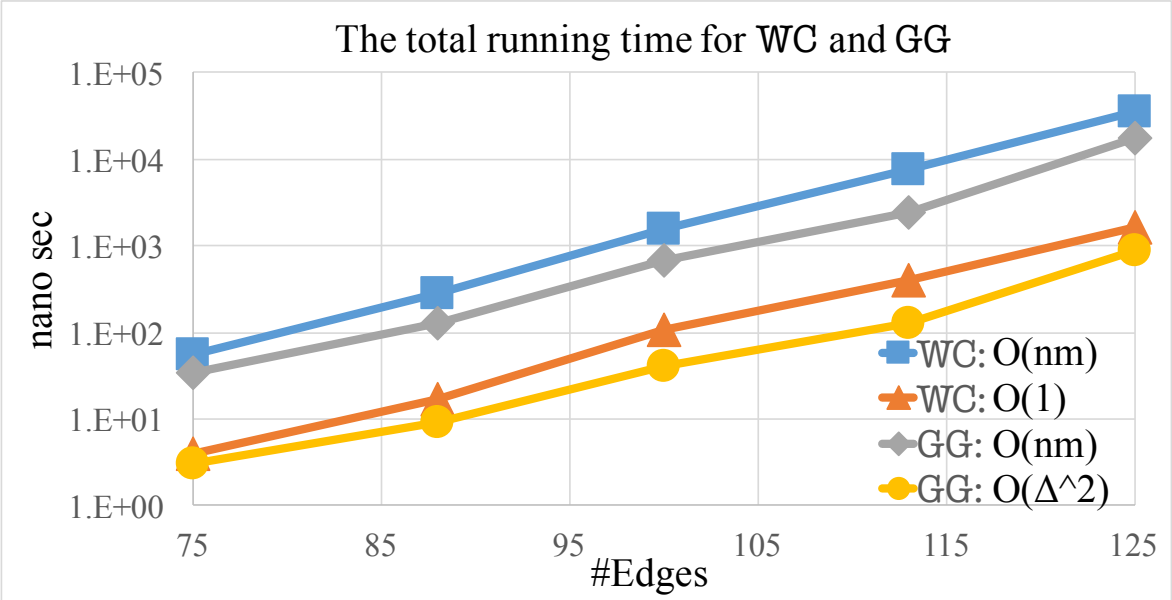
Proof. The correctness of `EIM` is obvious from Theorem 8. Next, we consider the time complexity of `EIM`. Let X be an iteration of `EIM`. In the preprocessing phase, `EIM` constructs $\mathcal{L}ist(G)$ in $O(|V| + |E|)$ time by using bucket sort. Next, we consider the

total time for deleting edges in an input graph. Each edge is deleted at most twice from Lemma 10 in each iteration. Moreover, from Lemma 17, the total number of deleted edges is $O(|\mathcal{T}|)$ in `EIM`. Hence, the total time of edge deletion is $O(|\mathcal{T}|)$ time since each edge can be removed in constant time from the input graph. Next, we consider the total time of the updating $\mathcal{L}ist(G(X))$. When `EIM` removes an edge $e = \{u, v\}$ from X, `EIM` moves $u \in L_i$ to L_{i-1} and $v \in L_j$ to L_{j-1}. Since it can be done in constant time, the time complexity of `EIM` is $O(|\mathcal{T}|)$. In addition, every iteration X in $\mathcal{T}$ has a child at least two. Hence, the number of solutions is $\Omega(|\mathcal{T}|)$. Therefore, `EIM` runs in $O(|\mathcal{T}| / |\mathcal{T}|) = O(1)$ time per solution. □

Experimental result

We conducted experiments on artificial data to evaluate the computational speed. We implemented `EIM`. The experimental environment is as follows: C++ with GCC7.3.0 and -O3 option. We considered the following graphs that were randomly generated: general graphs(`GG`), graphs without cycles with length four(`WC`). We experimented three algorithms, `EIM`, the simple polynomial delay algorithm, and a naive algorithm for `GG` and `WC`. The time complexity of the simple algorithm and the naive algorithm are amortized $O(nm)$ time and $O(2^n poly(n))$ time. The efficient algorithm is 200 times faster than the simple polynomial delay algorithm.

Although the efficient algorithms is theoretically $O(\Delta^2)$ for `GG`, its computation time was roughly equal to the computation time of `WC`. In all cases, we terminated the naive algorithm because the running time exceeds 20 minutes. This experiments show `EIM` is faster than simple algorithm.



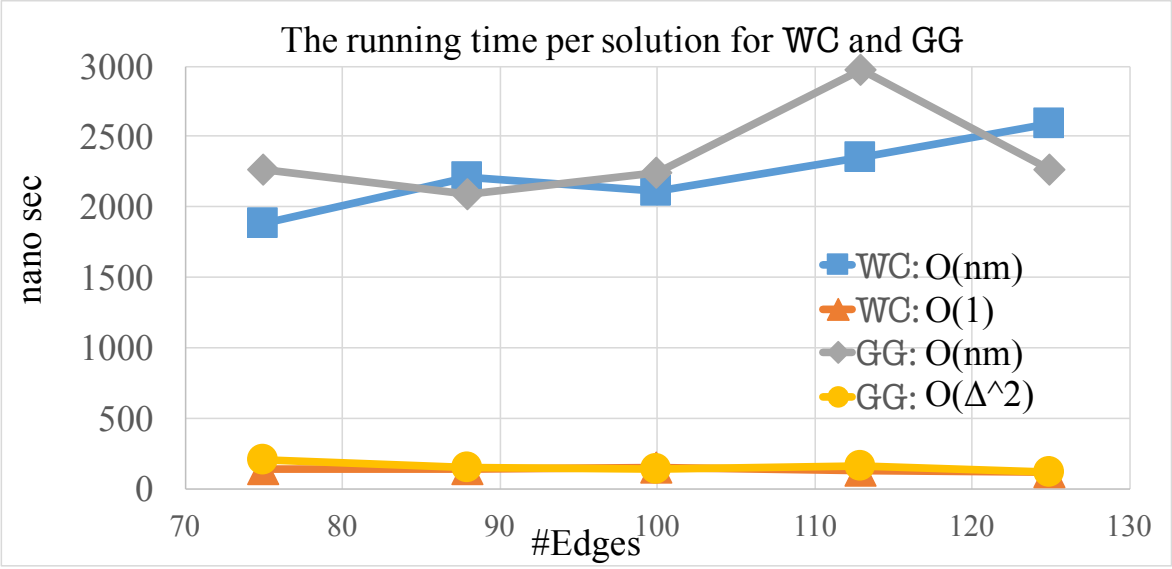


Figure 3.5: The total running time and the time per solution of induced matching enumeration. All input graphs in GG and WC have 30, 35, 40, 45, or 50 vertices. All cases, the average degree of the input graph is five.

#Edges	$O(1)$ for `WC`	$O(nm)$	naive
113	800,257	17,591,453	$2^{114} - 1 \simeq 2.0 \times 10^{34}$
125	7,689,450	119,371,878	$2^{126} - 1 \simeq 4.2 \times 10^{37}$

Table 3.1: The number of iterations in each induced matching enumeration. An input graph is in `WC`.

Chapter 4

Enumeration of Subgraphs with Bounded Girth

4.1 Introduction

In this chapter, we address enumeration of sparse subgraphs, that is, subgraphs with large girth. The *girth* of a directed or undirected graph G is the length of a shortest directed or undirected cycle. From the definition of girth, girth is ∞ if a graph has no cycle. We propose output sensitive algorithms for enumerating all subgraphs with girth at least g of a given graph G. Figure 4.1 shows some examples.

This problems generalize two well studied problems, i.e., enumeration of subtrees and induced subtrees [24, 60, 62, 75]. We consider enumeration of both *induced* and *edge* subgraphs (with girth g), and all algorithms can be easily adapted to relax the connectivity requirement or to deal with weighted graphs. We say that subgraphs whose girth is *at least* g simply as subgraphs with girth g. In addition, we generalize this problem to that of finding connected subgraphs with lower bounded *girth*.

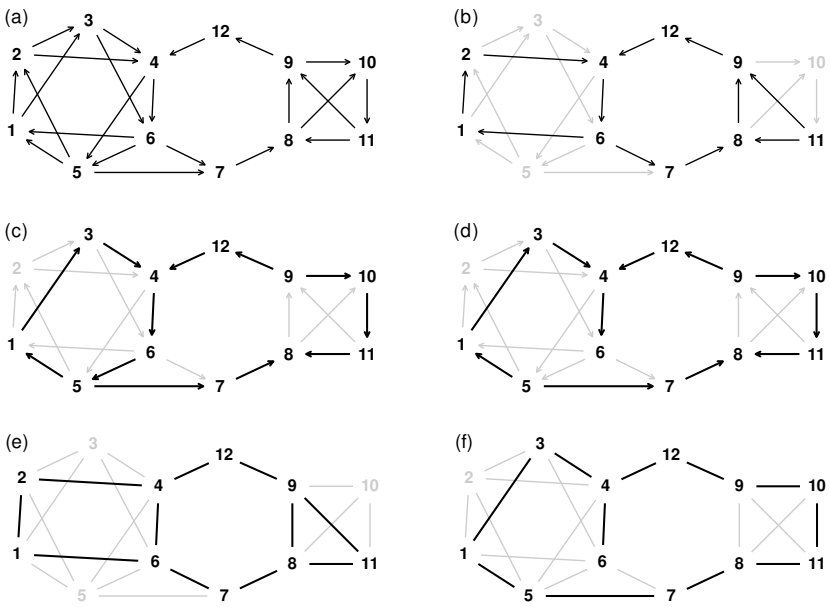


Figure 4.1: directed graph (a), an induced subgraph with girth 4 (b), an edge subgraph with girth 5 (c), and an acyclic edge subgraph (d). For the undirected case, an induced subgraph with girth 3 (e), and an edge subgraph with girth 10 (f). The subgraphs correspond to the vertices and edges in black.

Main result

We first consider the directed case. We propose two algorithms, `EDG-IS` and `EDG-ES`, for enumeration of *induced* and *edge* subgraphs with directed girth g. Both algorithms run in $O(n)$ time per solution using $O(n^3)$ space.

Next, we consider the undirected case and propose two algorithms, `EUG-IS` and `EUG-ES`. These algorithms enumerate *induced* and *edge* subgraphs with undirected girth g, respectively. All four algorithms are given for *connected* subgraphs, but can easily be applied to the enumeration of non-connected subgraphs and weighted graphs.

4.2 Algorithms for directed graphs

In this section, we propose two algorithms `EDG-IS` and `EDG-ES`. We first show a basic algorithm `EDG-Base`.

4.2.1 A basic algorithm for directed graphs

In the following, we describe the strategy of a basic algorithm for solving a problem. We define our problem in Problem 2.

Problem 2 directed girth g connected induced subgraph enumeration. *Enumerate all connected induced subgraphs S of a directed graph G with $g(S) \geq g$, without duplicates.*

We describe the detail of `EDG-Base` in Algorithm 2. This algorithm is based on binary partition. In the root iteration I, $S(I) = \emptyset$ since a subgraph made by a single vertex is acyclic, it have girth ∞ and every $v \in V$ is addible. Hence, $C(I)$ is equal to V in the first call of `RecEDG-Base`() performed by algorithm `EDG-Base`().

The recursive procedure can be seen as a form of binary partition. Let X be an iteration in an enumeration tree $\mathcal{T}$. We define the set of vertices $C(X) = \{v \in V \mid S(X) \cup \{v\}$ is connected and $g(S(X) \cup \{v\}) \geq k\}$, called *addible candidate set*. The vertices to be removed from the graph are represented by a set $R(X)$, and removed in the nested recursive calls. For a vertex $v \in C(X)$, we consider a set of solutions including $S(X) \cup \{v\}$ (Line 8). Next, we remove v from the graph and enumerate a set of solutions including $S(X)$ and not including v. Thus every cycle of the for loop can be seen as a binary partition step. However, grouping these steps in a single iteration is useful in the complexity analysis.

Algorithm 2: Enumerating all connected induced subgraphs with girth g in a directed graph G

```
1 Procedure EDG-Base(G, g)        // G: (directed) graph, g: positive integer
2     RecEDG-Base(∅, G, g);
3 Procedure RecEDG-Base(S, G, g)
4     Output S;
5     C ← {v ∈ V \ S | G[S ∪ {v}] is connected and has girth g};
6     R ← ∅;
7     for v ∈ C do
8         RecEDG-Base(S ∪ {v}, G \ R, g);    // find subgraphs containing v
9         R ← R ∪ {v};                       // find subgraphs not containing v
```

Correctness. Proving that each output of `EDG-Base` is an induced subgraph with girth at least g is trivial, since every vertex added to $S(X)$ has passed the check in Line 5. It is easy to prove that each output of `EDG-Base` has no duplication. All solutions found in sub-calls of Line 8 contain v. However, all solutions found during following cycles of the *for* loop will not contain the same v. Hence, there is no duplication. Finally, we show that `EDG-Base` outputs all solutions.

Lemma 19. *The algorithm* `EDG-Base` *outputs all solutions.*

Proof. We prove this by induction. We assume that `EDG-Base` outputs all solution with the size k. We consider a solution S^* with the size $k+1$. Let v be a vertex in S^*. From the assumption, `EDG-Base` outputs $S^* \setminus \{v\}$. Hence, there exist an iteration X which output $S^* \setminus \{v\}$. If a graph G in X has v, then we output S^*. Otherwise, we consider a path between the root iteration I to X. Let Y be an iteration which

remove a vertex v in Line 9. Hence, there is a sibling Z of Y which satisfies $S(Z) \subseteq S^*$ and G in Z includes all vertices in S^*. Thus, `EDG-Base` outputs S^* and the statement holds. □

Cost analysis. As every iteration will output a solution in Line 4, the cost per solution is clearly bounded by the cost of an iteration. This corresponds to the cost of computing $C(X)$ in Line 5. The trivial way to build $C(X)$ is to compute the girth of $G[S(X) \cup \{u\}]$ for each vertex $u \in V$. Since the girth can be computed in $O(nm)$ time [56], a total cost of $O(n^2m)$ per solution.

4.2.2 Improvement for induced subgraph enumeration

The bottleneck of `EDG-Base` is the girth computation. We improve this computation by using special distance matrices. In the following we show how modify this algorithm to obtain `EDG-IS`, which reduces the cost of `EDG-Base` by a factor $O(nm)$, obtaining $O(n)$ time per solution. First, consider the following fundamental property.

Observation 1. *Let $\ell(v, S(X))$ be the length of a shortest cycle containing v in $S(X)$. If, for a vertex $u \notin S(X)$, $\ell(v, S(X) \cup \{u\}) < \ell(v, S(X))$, then the shortest cycle containing v in $S(X) \cup \{u\}$ must contain u.*

As this applies to every $v \in S(X)$ and $u \notin S(X)$, this implies a more general property.

Observation 2. *If A is a subgraph of G with girth g, and $A \cup \{v\}$ has girth $g' < g$, the shortest cycle in $A \cup \{v\}$ must involve v.*

Let Y be an iteration of the parent of X, $S(Y)$ be the solution in Y, and $C(Y)$ be a candidate set. We have $S(Y) = S(X) \setminus \{v\}$ and $C(Y) = \{v \in V \setminus S(Y) \mid G[S(Y) \cup \{v\}]$

is connected and has girth $g\}$. Every addible vertex $v \in C(X)$ must either have been in $C(S(Y))$ as well, or be a neighbor of v. Otherwise the subgraph $G[S(X) \cup \{v\}]$ satisfies either $g(G[S(X) \cup \{v\}]) < g$ or be disconnected. We can use this properties to efficiently identify all vertices $v \in C(X)$. An addible candidate set $C(X)$ will be made of all the vertices in $C(Y)$ that still pass the check in Line 5 after adding v to $S(X)$, and all the vertices in $N(v) \setminus R(X)$ which were not already in $C(Y)$: these are only connected to $S(X)$ by v and hence cannot participate in any cycle.

We will also keep in each iteration a *special distance matrix* for $S(X)$, that is a matrix M_X of size $|C(X)| \times |C(X)|$ such that for each pair $x, y \in C(X)$, $M_X(x, y)$ is equal to the distance between x and y in the induced subgraph $G[S(X) \cup \{x, y\}]$. Clearly, if $M_X(x, y) + M_X(y, x) < g$ then $G[S(X) \cup \{x, y\}]$ has a cycle shorter than g. By using M_X and Observation 2, we can conclude the following.

Lemma 20. *Given $C(Y)$ the candidate set in the parent iteration, and the special distance matrix M_Y for $S(Y)$, Line 5 can be rewritten as follows:*

$$C(X) \leftarrow \{x \in C(Y) \mid M_Y(x, v) + M_Y(v, x) \geq g\} \cup (N(v) \setminus (R(X) \cup C(Y))) \quad (4.1)$$

With this technique, $C(X)$ is computed in $O(|C(Y)| + |N(v)|)$ time. After computing $C(X)$ we need to compute M_X, i.e., the special distance matrix for $S(X) = S(Y) \cup \{v\}$ which will be passed to the child iteration. To ease this we will use the M_X matrix built in the parent iteration, which we call M_Y: we must simply check if the shortest path between two vertices x and y has been improved by adding v to $S(X)$. In other words, given M_Y and $C(X)$, we can compute M_X in $O(|M_X|) = O(|C(X)|^2)$ time as for each $x, y \in C(X)$, $M_X(x, y) = \min(M_Y(x, y), M_Y(x, v) + M_Y(v, y))$.

As for the first iteration, with $S(I) = \emptyset$ and $C(I) = V$, M_I is computed in $O(|M_I|) = O(|C(I)|^2)$ time, since for each $x, y \in C(I)$, $M_I(x, y) = 1$ if $(x, y) \in E$,

and 0 otherwise. The improved algorithm `EDG-IS` is built by modifying iterations of `EDG-Base` as follows:

- The first iteration initializes M_I.
- The $C(X)$ and M_X are passed to child iterations as $C(Y)$ and M_Y.
- The $C(X)$ and M_X are built using $C(Y)$ and M_Y.
- Line 5 is modified as in Lemma 20.

We omit the revised pseudo-code, but one may look for reference at that of Algorithm 3 in the following section (which solves the *edge subgraph* version of the problem), as the structure is essentially the same and auxiliary data structures are shown.[1]

Cost analysis. Every iteration of `EDG-IS` will take $O(|C(Y)| + |C(X)|^2 + |N(v)|)$ time to compute $C(X)$ and M_X. While this is trivially bounded by $O(n^2)$, we show that it can be further improved by means of *amortized analysis*.

Let X be an iteration and Y be a parent of X. An iteration X has the sets $S(X)$, $R(X)$, $C(X)$, and the matrix M_X. Note that X have exactly $|C(X)|$ child iterations and take $O(|C(Y)| + |C(X)|^2)$ time to execute. However, X subdivide the $O(|C(X)|^2)$ portion of the cost equally among its $|C(X)|$ children, for a total of $|C(X)|$ each. Every iteration receive a $O(|C(X)|)$ cost, and be charged only from its parent of an additional $O(|C(X)|)$ time, for a total of $O(|C(X)|) = O(|N[S(X)]|)$ time per each iteration. Since $|N(v)| = O(|N[X]|)$, `EDG-IS` is amortized $O(|N[S(X)]|)$ time.

[1] In particular, note how rather than modifying G we simply pass the set X to children iterations to keep track of the removed nodes/edges.

Considering the space usage, $S(X)$, $C(X)$, and $R(X)$ may be efficiently stored by keeping track of just the difference between the parent and child iterations for an amortized space usage of $O(n)$. As for M_X, we do not actually need to compute a separate matrix in each iteration. We simply update the cells of M_Y and use M_Y as M_X, accessing only the cells corresponding to indices in $C(X)$. The depth of the enumeration tree, i.e., the number of changes we need to keep track of, is $O(|S(X)|)$; the total space usage will thus be $O(|M_X| \cdot |S(X)|) = O(|N[S(X)]|^2 \cdot |S(X)|) = O(n^3)$.

As g does not impact the cost, `EDG-IS` can enumerate acyclic subgraphs, i.e., subgraphs with girth at least $n+1$, in $O(n)$ time per solution as well. Furthermore, distance is meaningless in acyclic subgraphs, as we only care about whether x *can reach* y or not. Each cell of M_X is updated at most once, for a total space usage of $O(|M_X|) = O(|N[S(X)]|^2) = O(n^2)$. We can finally state the correctness and complexity of `EDG-IS`.

Theorem 21. `EDG-IS` *enumerates the induced subgraphs of a graph G with girth at least g exactly once in $O(\sum_{S \in \mathcal{S}} |N[S]|)$ total time and $O(\max_{S \in \mathcal{S}}\{|N[S]|^2\} \cdot |S|) = O(n^3)$ space, where $\mathcal{S}$ is the set of solutions.*

Theorem 22. `EDG-IS` *enumerates the acyclic induced subgraphs of a graph G exactly once in $O(\sum_{S \in \mathcal{S}} |N[S]|)$ total time and $O(\max_{S \in \mathcal{S}}\{|N[S]|^2\}) = O(n^2)$ space, where $\mathcal{S}$ is the set of solutions.*

Weighted and non-connected cases. `EDG-IS` is given for unweighted graphs. However, it should be remarked that it is trivially adapted to *weighted* graphs, where the weight of a cycle is the sum of its edges, and the girth is the smallest weight of a cycle. This is done by simply initializing $M_X(x, y)$ in the first iteration to the weight of the edge (x, y), rather than 1, as long as $g > 0$. The approach works in the presence

of negative edges and even negative cycles, as a negative cycle can never be added to $S(X)$ since it would cause $g < 0$. Furthermore, `EDG-IS` can be trivially modified to drop the connectivity constraint by simply setting $C(X) = V$ in the first iteration, so that every vertex can be immediately considered for addition (we can then also ignore the addition of vertices in $N(v)$ to $C(X)$, see Lemma 20). In this way all the induced subgraphs with girth g will be enumerated, rather than just the connected ones.

Similar trivial adaptations to cover the weighted and non-connected case are possible for all the other algorithms proposed in the chapter for both induced and edge subgraphs.

4.2.3 Improvement for subgraph enumeration

In this section we describe an algorithm for enumeratng all *edge* subgraphs with girth at least g, or more formally, to solve Problem 3.

Problem 3 directed girth g connected subgraph enumeration. *Enumerate all connected subgraphs S of a directed graph G with $g(S) \geq g$, without duplicates.*

The algorithm `EDG-ES` is detailed in Algorithm 3. Firstly, in an iteration X, the solution $S(X)$, the set of addible candidates $C(X)$, and removed elements $R(X)$ are sets of *edges* rather than vertices. Secondly, the size of candidate edges is an important in the complexity of `EDG-ES`. Furthermore, we show the auxiliary data structures explicitly, to aid the understanding of the cost analysis.

We consider the current solution in an iteration X. Let $S(X)$ be a solution corresponding to an iteration X, a set $R(X)$ of removed edges, and the set $C(X) \subseteq E \setminus (S(X) \cup R(X))$ of edges that are addible to $S(X)$. In addition, we will subdivide $C(X)$ into $C^{\text{in}}(X)$ and $C^{\text{ex}}(X)$. Let $S_N(X)$ be the set of vertices incident to edges

in $S(X)$, $\forall e = \{x, y\} \in C(X)$, $e \in C^{\text{in}}(X)$ if $\{x, y\} \subseteq S_N(X)$, and $e \in C^{\text{ex}}(X)$ otherwise. Again, we find all solutions in a binary partition fashion by selecting an edge $e \in C(X)$, and first considering all subgraphs with contain $S(X) \cup \{e\}$, then removing e from $C(X)$ and considering those that contain $S(X)$ but not e.

We call this algorithm `EDG-ES`, and we can reconstruct its structure by simple modifications of `EDG-IS`. Each cycle of the *for* loop in Line 7 in Algorithm 2 considers an edge $e \in C^{\text{in}}(X)$ rather than a vertex. Furthermore, the updated $C(X)$ (Line 5) should be computed as $C(X) \leftarrow \{e' \in E \setminus (S(X) \cup R(X)) \mid \text{and } G' = (V[S(X) \cup \{e'\}], S \cup \{e'\})$ is connected has girth $g\}$. Finally, `EDG-ES` select e from $C^{\text{ex}}(X)$ *only if* $C^{\text{in}}(X) = \emptyset$. For brevity, we omit the correctness proof which consists in simply retracing that of `EDG-IS`.

Again, we employ a *special distance matrix* M_X. Let $C_N(X)$ be the set of vertices incident to at least one edge in $C(X)$. In this case, the size of M_X is $|C_N(X)| \times |C_N(X)|$, and for each pair $x, y \in C_N(X)$, $M_X(x, y)$ is equal to the distance between x and y in the edge subgraph $G' = (V[S(X)], S(X))$. For two edges $e_1 = (x, y)$ and $e_2 = (w, z)$ in $C(X)$, if there is a cycle shorter than g in $G'' = (V[S(X) \cup \{e_1, e_2\}], S(X) \cup \{e_1, e_2\})$, then we will have $M_X(y, w) + M_X(z, x) + 2 < g$, since any cycle involving e_1 and e_2 must traverse the vertices y, w, z, and x in this order. After adding $e = \{a, b\}$ to $S(X)$, the edges in $N^e(a) \cup N^e(b)$ but not in X may enter $C(X)$, which can thus be computed similarly to how done in Lemma 20 for the induced case, i.e.

$$C(X) \leftarrow \{e' = \{c, d\} \in C(Y) \cup (N^e(a) \cup N^e(b)) \setminus R(X) \mid M_X(b, c) + M_X(a, d) + 2 \geq g\}, \tag{4.2}$$

where Y is a parent iteration of X and the 2 is added to account using the edges e and e' and can be replaced by their weight for the weighted case. The values of M_X can be updated after adding $e = \{a, b\} \in C(X)$ to $S(X)$. We have that $M_X(y, w)$, i.e., the

distance "from" e_1 to e_2 in $G' = (V[S(X) \cup \{e\}], S(X) \cup \{e\})$, was either improved by using e or is unchanged. That is $M_X(y, w) = \min(M_X(y, w), M_Y(y, a) + M_Y(b, z) + 1)$, where the 1 is added to account for using e. Note that we replaced by the weight of e when weighted case. Thus, `EDG-ES` will also follow the structure in Algorithm 2, modifying iterations of `EDG-Base` as follows:

- The first iteration initializes M_X.
- The sets $C^{\text{in}}(X)$, $C^{\text{ex}}(X)$, $C_N(X)$, $S_N(X)$ and M_X are passed to child iterations.
- $C(X)$, $C_N(X)$, $S_N(X)$ and M_X are updated using those passed from the father iteration.
- The candidates in $C^{\text{ex}}(X)$ will be selected only after $C^{\text{in}}(X)$ is empty.
- Line 9 is modified as in Equation (4.2).

Cost analysis. By implementing the updates in Line 9 similarly to how done in `EDG-IS`, and performing the same amortized analysis, one could easily find that `EDG-ES` has a complexity of $O(m)$ time per solution, which is a factor $O(mn)$ faster than the baseline. In the following, however, we will further reduce the cost of Line 9 and obtain $O(n)$ time per solution.

In particular, let us focus on the update of the $C^{\text{in}}(X)$ and $C^{\text{ex}}(X)$ sets. When `EDG-ES` selects $e \in C^{\text{ex}}(X)$, updating the sets can trivially be done in $O(m)$ time by testing each edge $f \in E \setminus (S(X) \cup R(X))$ with M_X as in Equation 4.2. However, this can be simplified by means of the following:

Algorithm 3: Enumerating all connected edge subgraphs with girth g in a directed graph G

```
1  Procedure EDG-ES(G = (V, E), g)
2      R ← ∅;
3      foreach e = {x, y} ∈ E do
4          RecEDG-ES(∅, ∅, ∅, ∅, ∅, ∅, R, e, g));
5          X ← X ∪ {v};
6  Procedure RecEDG-ES(S, C^in, C^ex, C_N, S_N, M, R, e, g)
       // let e = {a, b}
7      S, S_N ← S ∪ {e}, S_N ∪ {a, b};
8      Output S;
9      Update C_N, C^in, C^ex, M for the new S and R;
10     for f ∈ C^in do
11         RecEDG-ES(S, C^in, C^ex, C_N, S_N, M, R, f, g) ; // subgraphs containing f
12         R ← R ∪ {f} ;                                    // subgraphs not containing f
13     for f ∈ C^ex do
14         RecEDG-ES(S, C^in, C^ex, C_N, S_N, M, R, f, g) ; // subgraphs containing f
15         R ← R ∪ {f} ;                                    // subgraphs not containing f
16     S ← S \ {v}; R ← R \ C ;                             // restore S and R
17     Restore C_N, C^in, C^ex, M for the restored S and X ;
```

Lemma 23. *Let $e = \{a, b\} \in C^{\text{ex}}(X)$ be the edge selected and added to $S(X)$ by* EDG-ES*, with $C^{\text{in}}(X) = \emptyset$. Without loss of generality let $a \in S_N(X)$ and $b \notin S_N(X)$. Then*

- *The updated $C^{\text{in}}(X)$ is contained $N^e(b) \setminus (S(X) \cup R(X))$.*

- *The updated* $C^{\text{ex}}(X)$ *is contained in* $C^{\text{ex}}(X) \cup N^e(b) \setminus (S(X) \cup R(X))$.
- *Both* $C^{\text{in}}(X)$ *and* $C^{\text{ex}}(X)$ *can be updated in* $O(\Delta)$ *time.*

Proof. Since b is the only new vertex in $S_N(X)$, the new edges in $C^{\text{in}}(X)$ and $C^{\text{ex}}(X)$ must be adjacent to b. Any new edge in $C^{\text{in}}(X)$ must be removed from $C^{\text{ex}}(X)$. $C^{\text{in}}(X)$ and $C^{\text{ex}}(X)$ can be computed by scanning $N^e(b)$ and testing each edge $\{b, x\}$ not in $S(X)$ or $R(X)$ in constant time using M_X, adding the edges that pass the girth test to $C^{\text{in}}(X)$ if $x \in S_N(X)$ and to $C^{\text{ex}}(X)$ otherwise. This takes $O(\Delta)$ time. □

Note that `EDG-ES` only selects e from $C^{\text{ex}}(X)$ once $C^{\text{in}}(X)$ is empty, and otherwise it will select it from $C^{\text{in}}(X)$. Selecting e from $C^{\text{in}}(X)$ always decreases $|C^{\text{in}}(X)|$ by at least 1: indeed no new edge may enter $C^{\text{in}}(X)$ since $S_N(X)$ is unchanged, but e itself is removed. When $C^{\text{in}}(X)$ is empty and we select e from $C^{\text{ex}}(X)$, $|C^{\text{in}}(X)|$ become at most Δ (Lemma 23). We state that

Lemma 24. *For any iteration* X, $|C^{\text{in}}(X)| \leq \Delta$.

When `EDG-ES` selects the edge e from $C^{\text{in}}(X)$, From Lemma 24, we can also update $C^{\text{in}}(X)$ and $C^{\text{ex}}(X)$ faster than in $O(m)$ time:

Lemma 25. *Let* X *be an iteration and* $e = \{a, b\} \in C^{\text{in}}(X)$ *be the edge selected and added to* $S(X)$. *Then the updated* $C^{\text{in}}(X)$ *is included in* $C^{\text{in}}(X) \setminus \{e\}$ *and can be computed in* $O(\Delta)$, *and* $C^{\text{ex}}(X)$ *is unchanged.*

Proof. As $S_N(X)$ is unchanged, no edge enters $C^{\text{in}}(X)$, but e is removed. Whether each edge remains in $C^{\text{in}}(X)$ can be tested in constant time using M_X, which takes $O(\Delta)$ time as $|C^{\text{in}}(X)| \leq \Delta$ by Lemma 24. Finally, as every edge in $C^{\text{ex}}(X)$ still exactly one extreme in $S_N(X)$, it may not participate in any cycle in $G(V[S_N(X)], S(X))$, and since $S_N(X)$ is unchanged no edge is either removed from or added to $C^{\text{ex}}(X)$. □

We can now proceed to give the complexity `EDG-ES`. We consider any iteration X, with its sets $S(X)$, $R(X)$, $C^{\text{in}}(X)$, $C^{\text{ex}}(X)$, $C_N(X)$ and the matrix M_X as computed in Line 9, and Z, a child iteration of X (performed in either Line 11 or 14) with its sets $S(Z)$, $R(Z)$, $C^{\text{in}}(Z)$, $C^{\text{ex}}(Z)$, $C_N(Z)$ and the matrix M_Z.

From Lemmas 25 and 23, we can update $C^{\text{in}}(X)$ and $C^{\text{ex}}(X)$ to obtain $C^{\text{in}}(Z)$ and $C^{\text{ex}}(Z)$ in $O(\Delta)$ time. Furthermore, $C_N(Z)$ can be obtained in constant time, and using M_X and $C_N(Z)$ we can update M_X to obtain M_Z in $O(|C_N^{\text{in}}(Z) \cup (C_N^{\text{ex}}(Z) \cap S_N(Z))|^2)$ time. The total cost of Line 9 will thus be $O(\Delta + |C_N^{\text{in}}(Z) \cup (C_N^{\text{ex}}(Z) \cap S_N(Z))|^2)$. Since $O(\Delta) = O(|S_N(Z) \cup C_N^{\text{ex}}(Z)|)$, the total cost of Line 9 is $O(|C_N^{\text{in}}(Z) \cup (C_N^{\text{ex}}(Z) \cap S_N(Z))|^2)$.

However, we have that for each edge in $C^{\text{in}}(Z)$ and $C^{\text{ex}}(Z)$ there are two vertices in $C_N^{\text{in}}(Z)$ and $C_N^{\text{ex}}(Z)$ respectively, which means $|C_N^{\text{in}}(Z)| + |C_N^{\text{ex}}(Z)| \leq 2(|C^{\text{in}}(Z)| + |C^{\text{ex}}(Z)|)$. As Z has $|C^{\text{in}}(Z)| + |C^{\text{ex}}(Z)|$ children iterations, we can give the same amortized analysis as for `EDG-IS`. An iteration Z will subdivide equally among its children the $O(|C_N^{\text{in}}(Z) \cup (C_N^{\text{ex}}(Z) \cap S_N(Z))|^2)$ time component of its cost, for a total of $O(|C_N^{\text{in}}(Z) \cup (C_N^{\text{ex}}(Z) \cap S_N(Z))|) = O(|S_N(Z)|) = O(n)$ for each child.

The space complexity of `EDG-ES` is dominated by the space needed to store $S(X)$, $C^{\text{in}}(X)$, $C^{\text{ex}}(X)$ and $R(X)$. It can be stored in amortized $O(m)$ space by keeping track of the differences between parent and children iterations. Next, $C_N(X)$ and $S_N(X)$ can be stored in $O(n)$ space. Finally, M_X has $O(\max_{S \in \mathcal{S}}\{|S_N|^2\})$ cells. For each cell we keep track of at most n changes, where $\mathcal{S}$ is the set of solutions. Indeed, while the depth of the recursion is bounded by m, each value $M_X(i,j)$ corresponds to a distance between two vertices i and j, which is bounded by n. Hence, a total space usage is bounded by $O(n \cdot \max_{S \in \mathcal{S}}\{|S_N|^2\}) = O(n^3)$[2].

[2]This is different in the weighted case, in which distances can be reduced by less than 1, and will

Algorithm 4: Enumerate all connected induced subgraphs with girth g.

```
Procedure EUG-Base(G, g)    // G: (undirected) graph, g: positive integer
    RecEUG-ES(∅, G, g);
Procedure RecEUG-ES(S, G, g)    // S: the current solution
    Output S;
    R ← ∅;
    C(S) ← {x ∈ V(G) \ S | G[S ∪ {x}] is connected and has girth g};
    for v ∈ C(S) do
        RecEUG-ES(S ∪ {v}, G \ R, g);
        R ← R ∪ {v};
    return;
```

As for acyclic edge subgraphs, there are only two possible values for $M_X(i,j)$. As we only need to keep track of one update, the space usage will be $O(\max_{S\in\mathcal{S}}\{|S_N|^2\}) = O(n^2)$. We can finally state the cost of EDG-ES.

Theorem 26. *Given a graph $G = (V, E)$,* EDG-ES *enumerates the edge subgraphs of G with girth at least g exactly once in $O(\sum_{S\in\mathcal{S}} |S_N|)$ total time space $O(n \cdot \max_{S\in\mathcal{S}}\{|S_N|^2\})$.*

Theorem 27. *Given a graph $G = (V, E)$,* EDG-ES *enumerates the edge subgraphs of G with girth at least g exactly once in $O(\sum_{S\in\mathcal{S}} |S_N|)$ total time space $O(\max_{S\in\mathcal{S}}\{|S_N|^2\})$.*

4.3 Algorithms for undirected graphs

In the following sections, we consider the enumeration problems corresponding to those above, for the case of undirected graphs.

Problem 4 undirected girth g connected induced subgraph enumeration. *Enumerate all connected induced subgraphs S of an undirected graph G with $g(S) \geq g$, without duplicates.*

Problem 5 undirected girth g connected subgraph enumeration. *Enumerate all connected subgraphs S of an undirected graph G with $g(S) \geq g$, without duplicates.*

To solve these problems, we will show two algorithms with a similar structure to those above, but which require significantly different techniques to achieve efficiency.

4.3.1 A basic algorithm for undirected graphs

Algorithm `EUG-Base`, detailed in Algorithm 4, represents a basic strategy for Problem 4, much like that of Algorithm 2. Again, while `EUG-Base` enumerates solutions by picking vertices on each iteration, we can obtain an enumeration algorithm for Problem 5 by modifying `EUG-Base` so that it picks edges instead.

Let G, X, and $S(X)$ be respectively the undirected graph in input, an iteration of the algorithm, and the solution that the iteration X received from its parent iteration. The set of *candidate vertices* for $S(X)$ is defined as follows: $C(X) = \{v \in V \setminus S(X) \mid g(S(X) \cup \{v\}) \geq g$ and $S(X) \cup \{v\}$ is connected. $\}$, meaning that $S(X) \cup \{v\}$ is a solution. We recall that that iterations of the execution on a graph G correspond to nodes of the *enumeration tree* $\mathcal{T}$, where each nested iteration of a node corresponds to a child node in $\mathcal{T}$.

thus require using $O(n^2m)$ space.

Finally, the correctness proof for Algorithm 4 can be obtained similarly to that for Algorithm 2, and using Itai's algorithm [32] to compute the girth of a graph in $O(mn)$, we can obtain a first trivial complexity bound.

Theorem 28. `EUG-Base` *solves Problem 4, i.e., enumerates all connected induced subgraphs with girth g of a graph without duplication, with delay $O(n^2m)$.*

Proof. By the same reasoning as Algorithm 2, `EUG-Base` enumerates all solutions without duplication.

As for its delay, since every iteration outputs a solution, it is sufficient to bound the time complexity of one iteration. The bottleneck of `RecEUG-ES` is Line 7: in order to compute $C(X)$, `EUG-Base` must iterate over all vertices $v \in V$ and check whether the girth of $G[S(X) \cup \{v\}]$ is g.

By using time Itai's algorithm [32], we can test each v in $O(nm)$, thus the total cost is bounded by $|V| \cdot O(nm) = O(n^2m)$. □

4.3.2 Improvement for induced subgraph enumeration

The bottleneck of `EUG-Base` is the computation of the candidate set. In this section, we present a more efficient algorithm `EUG-IS` for Problem 4. `EUG-IS` is based on `EUG-Base`, but each iteration exploits information from the parent iteration, and maintains distances in order to improve the computation of the candidate set. The procedure is shown in Algorithm 5.

Let π be a shortest path between u and v. We call a path which is a shortest u-v path without π a *second shortest path*. If there is shortest paths two or more, then the length of a shortest path and a second shortest path is same.

`EUG-IS` uses the second distance between vertices defined as follows. Let u and

Algorithm 5: Updating data structures in `EUG-IS`.

```
1  Procedure NextCand(v, C, M^u1, M^u2, S, g, G)
2      C ← UpdateCand(v, S);
3      M^u1, M^u2 ← UpdateU1(v, C);
4  Function UpdateCand(v, S, C)
5      C ← N(v) ∪ C;
6      foreach u ∈ C do
7          if M^u1(u, v) + M^u2(u, v) ≥ g then  C ← C ∪ {u} ;
8      return C;
9  Function UpdateU1(v, C)
10     foreach u ∈ C ∪ S, w ∈ C do
11         M^u2(u, w) ← min{max{M^u1(u, w), M^u1(u, v, w)}, M^u2(u, w), M^u2(u, v) +
             M^u1(v, w), M^u1(u, v) + M^u2(v, w)}. ;
12         M^u1(u, w) ← min{M^u1(u, w), M^u1(u, v, w)};
13     return M^u1, M^u2
```

v be vertices in $C(X)$. We denote by $M_X^{\mathtt{u1}}(u,v)$ the distance between v and u in $G[S(X) \cup \{v,u\}]$, and by $M_X^{\mathtt{u2}}(u,v)$ the length of a second shortest path between u and v in $G[S(X) \cup \{u,v\}]$. Note that for any vertices $x, y \in G \setminus C(X)$, $M_X^{\mathtt{u1}}(x,y) = \infty$ and $M_X^{\mathtt{u2}}(x,y) = \infty$. Especially, we call $M_X^{\mathtt{u2}}(u,v)$ the *second distance* between u and v in $G[S(X) \cup \{u,v\}]$. Moreover, we write $M_X^{\mathtt{u1}}(u,w,v)$ and $M_X^{\mathtt{u2}}(u,w,v)$ for the distance and the second distance from u to v via a vertex w, respectively. Let π and π' be respectively a v-u shortest path and a v-u second shortest path. Since π and π' share u and v, and there is a vertex x such that $x \in \pi$ and $x \notin \pi'$ holds, H must have a cycle including v and u, where H is a subgraph of G such that $V(H) = V(\pi) \cup V(\pi')$ and

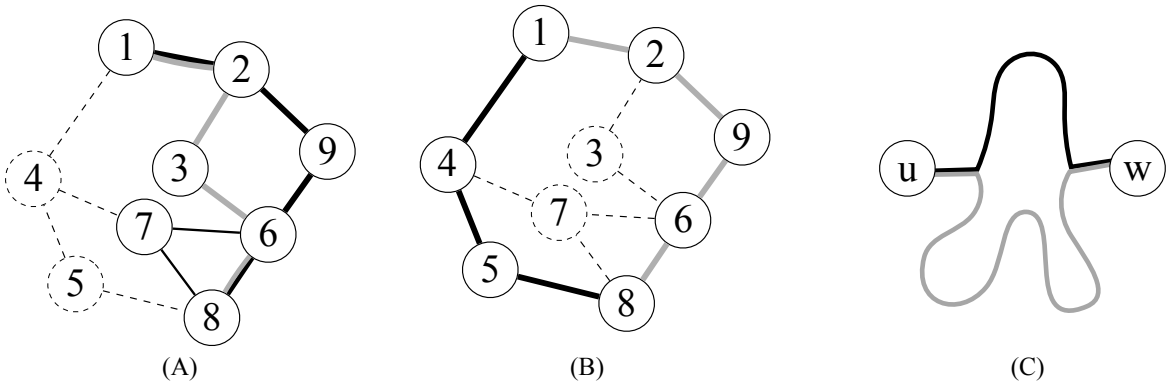


Figure 4.2: An example of shortest path and second shortest path.

$E(H) = E(\pi) \cup E(\pi')$. Figure 4.2 shows an example of a cycle made by π and π'. Let $S(A) = \{1, 2, 3, 6, 7, 8, 9\}$ and $S(B) = \{1, 2, 4, 5, 6, 8, 9\}$. Dashed edges and vertices are not contained by induced subgraphs. Black and gray paths show respectively shortest and second shortest paths. When $u = 1$, $v = 8$, $M_A^{\mathtt{u1}}(u, v) = 4$ and $M_A^{\mathtt{u2}}(u, v) = 4$. Similarly, $M_B^{\mathtt{u1}}(u, v) = 3$ and $M_B^{\mathtt{u2}}(u, v) = 4$ hold in $G[S(B)]$. To compute the candidate set efficiently, we will use the following lemmas. In the following lemmas, let Y and $S(Y)$ be the parent iteration of X and a solution of Y. Moreover, v be a vertex in $C(Y)$ such that $S(X) = S(Y) \cup \{v\}$.

Lemma 29. *Let u and w be two vertices in $C(Y)$ and $g = g(G[S(Y)])$. (A) $g(G[S(Y) \cup \{u, w\}]) \geq g$ if and only if (B) $M_Y^{\mathtt{u1}}(u, w) + M_Y^{\mathtt{u2}}(u, w) \geq g$.*

Proof. Clearly, (A) $\rightarrow$ (B) holds by definition of $M_Y^{\mathtt{u1}}$ and $M_Y^{\mathtt{u2}}$. For the direction (B) $\rightarrow$ (A), consider a shortest cycle C in $G[S(Y) \cup \{u, w\}])$ in the following three cases: (I) $u, w \notin C$: $|C| \geq g$ since $g(G[S(Y)]) \geq g$. (II) Either u or w in C: $|C| \geq g$ since u and w belong to $C(Y)$. (III) Both u and w in C: C can be decomposed into two u-w paths π_1 and π_2. Without loss of generality, $|\pi_1| \leq |\pi_2|$. If π_1 is a u-w shortest path, then $|C| \geq g$ from (B), since π_2 is at least as long as the *second distance* $M_Y^{\mathtt{u2}}(u, w)$.

Otherwise, there is a u-w shortest path π_3 and a cycle C' consisting of a part of π_1 (or π_2) and a part of π_3. If C' contains w, then $|C'| = |C| \geq g$ since C is a shortest cycle. If C' does not contain w, then $|C'|$ is a cycle in $G[S(Y) \cup \{u\}]$, thus $|C'| \geq g$ because $u \in C(Y)$. □

Lemma 30. `EUG-IS` *computes $C(X)$ in $O(|C(Y)| + |N(v)|)$ time.*

Proof. From Lemma 29, vertex u in $C(Y)$ belongs to $C(X)$ if and only if $M_Y^{\mathtt{u1}}(u,v) + M_Y^{\mathtt{u2}}(u,v) \geq g$. This can be done in constant time. In addition, from the connectivity of $G[S(X)]$, $C(X) \setminus C(Y) \subseteq N(v)$. Thus, we can find $C(X) \setminus C(Y)$ in $O(|C(Y)| + |N(v)|)$ time. □

Next, we consider how to update the values of $M_X^{\mathtt{u1}}$ and $M_X^{\mathtt{u2}}$ when adding v to Y. We can update the old distances to the ones after adding v as in the Floyd-Warshall algorithm (see Algorithm 5), meaning that we can compute $M_X^{\mathtt{u1}}$ in $O\left(|C(Y)|^2\right)$ time. By the following lemma, the values of $M_X^{\mathtt{u2}}$ can be updated in constant time for each pair of vertices in $C(X)$.

Lemma 31. *Let u and w be vertices in $C(Y)$. $M_X^{\mathtt{u2}}(u,w)$ is the minimum value of the followings: (I) $\max\{M_Y^{\mathtt{u1}}(u,w), M_Y^{\mathtt{u1}}(u,v,w)\}$, (II) $M_Y^{\mathtt{u2}}(u,w)$, (III) $M_Y^{\mathtt{u2}}(u,v) + M_Y^{\mathtt{u1}}(v,w)$, and (IV) $M_Y^{\mathtt{u1}}(u,v) + M_Y^{\mathtt{u2}}(v,w)$.*

Proof. We consider a u-w shortest path π_1. $|P|$ is equal to $\min\{M_Y^{\mathtt{u1}}(u,w), M_Y^{\mathtt{u1}}(u,v,w)\}$. We assume that $|\pi_1| = M_Y^{\mathtt{u1}}(u,w)$. Let π_2 be a second shortest path in $G[X]$. If π_2 includes v, then $|\pi_2| = M_Y^{\mathtt{u1}}(u,v,w)$. Otherwise, $|\pi_2| = M_Y^{\mathtt{u2}}(u,w)$. In this case, $|\pi_2| = \min\{M_Y^{\mathtt{u2}}(u,w), M_Y^{\mathtt{u1}}(u,v,w)\}$. We assume that $|\pi_1| = M_Y^{\mathtt{u1}}(u,v,w)$. Let π_2 be a second shortest path π_2. If π_2 includes v, then $|\pi_2|$ is equal to $\min\{M_Y^{\mathtt{u1}}(u,v) + M_Y^{\mathtt{u2}}(v,w), M_Y^{\mathtt{u2}}(u,v) + M_Y^{\mathtt{u1}}(v,w)\}$. Otherwise, $|\pi_2| = M_Y^{\mathtt{u1}}(u,w)$. In this case, $|\pi_2| =$

$\min\{M_Y^{\mathtt{u1}}(u,w), M_Y^{\mathtt{u1}}(u,v) + M_Y^{\mathtt{u2}}(v,w), M_Y^{\mathtt{u2}}(u,v) + M_Y^{\mathtt{u1}}(v,w)\}$. Hence, the statement holds. □

From Lemma 31, we can analyze the time complexity of computing $M_X^{\mathtt{u2}}$.

Lemma 32. *We can compute $M_X^{\mathtt{u2}}$ from $M_Y^{\mathtt{u2}}$ and $M_Y^{\mathtt{u1}}$ in $O\left(|C(X)|^2\right)$ time.*

Theorem 33. `EUG-IS` *enumerates all solutions in $O\left(\sum_{S\in\mathcal{S}} |N[S]|\right)$ time using $O\left(n \cdot \max_{S\in\mathcal{S}}\{|N[S]|^2\}\right)$ space, where $\mathcal{S}$ is the set of all solutions.*

Proof. The correctness of `EUG-IS` follows from the fact that it performs the same operations as Algorithm 4, which is correct. We first consider the space complexity. In an iteration X, `EUG-IS` uses $O\left(|C(X)|^2\right)$ space for storing values of $M^{\mathtt{u1}}$ and $M^{\mathtt{u2}}$. In addition, the height of $\mathcal{T}$ is at most n. Therefore, `EUG-IS` uses $O\left(n \cdot \max_{S\in\mathcal{S}}\{|N[S]|^2\}\right)$ space.

Let c_X be $|C(X)|$ and $T(Y,X)$ be the time needed to generate X from Y, i.e., an execution of `NextCand`() (Algorithm 5). From Lemma 30, Lemma 31, and the Floyd-Warshall algorithm, $T(Y,X)$ is $O(c_Y + |N(v)| + c_X^2)$ time. Thus, by distributing the $O(|N(S(X))|)$ time from X to children of X, each iteration needs $O(|N(S(X))| + |N(v)|) = O(|N[S(X)]|)$ time since each iteration receives costs only from the parent and the grandparent. In addition, each iteration outputs a solution, and hence the total time is $O\left(\sum_{S\in\mathcal{S}} |N[S]|\right)$. □

4.3.3 Improvement for subgraph enumeration

We propose an algorithm, `EUG-ES`, for enumerating all subgraphs with girth g in an undirected graph G, detailed in Algorithm 6. A trivial adaptation of `EUG-IS` would run in $O(m)$ time per solution, as the candidate sets are sets of edges, whose size is

Figure 4.3: Black solid lines and gray solid lines represent inner edges and external edges, respectively.

$O(m)$. To improve this running time, `EUG-ES` selects candidates in a certain order, so that the number of candidate edges does not exceed no more than the number of nodes in the previous solution $G[S]$.

Let X be an iteration and $S(X)$ be the current solution. Note that $S(X)$ is an edge set. We first define an inner edge and an external edge as follows: an edge $e = \{u, v\}$ is an *inner edge* for $S(X)$ if $u, v \in G[S(X)]$, and an *external edge* otherwise (see Figure 4.3). Our main strategy is to reduce the number of inner edges in `EUG-IS`. We first consider the case when `EUG-ES` picks an external edge. In the following lemmas, let X be an iteration in enumeration tree $\mathcal{T}$, $S(X)$ be a solution in an iteration X, $C^{\text{in}}(X)$ and $C^{\text{ex}}(X)$ be a set of inner edges and external edges in $C(X)$, e be an edge in $S(X)$, and Y be the parent iteration of X satisfying $S(X) = S(Y) \cup \{e\}$. Here, $S_N(X)$ be the set of vertices incident to edges in S_R.

Lemma 34. *Let $e = \{x, y\}$ be an external edge such that $x \in S_N(X)$. Then $C(X) \subseteq (C(Y) \cup E(y)) \setminus \{e\}$, where $E(y)$ are the edges incident to y.*

Proof. An edge $g \notin E(y) \cup C(Y)$ may not be added to $S(X)$ as the resulting subgraph would be disconnected, and $e \notin C(X)$ since $e \in S(X)$. □

From Lemma 34, `EUG-ES` manages the candidate set $C(X)$ in $O(|C(X)| + |S_N(X)|)$ time when `EUG-ES` picks an external edge e since we can add all edges $e' \notin S(Y) \cup C(Y)$

incident to y and $S(X) \cup \{e'\}$ is a solution. Moreover, removed edges are at most $|S_N(Y)|$ since all removed edges have a vertex in $S_N(Y)$. In this case, `EUG-ES` can obtain $C^{\text{in}}(X)$ and $C^{\text{ex}}(X)$ in $O(|S(Y)|)$ time and $O(|C(X)|)$ time, respectively. Next, we consider that when `EUG-ES` picks an inner edge e. When we pick an inner edge, $C(X)$ is monotonically decreasing.

Lemma 35. *If e is an inner edge, then $C^{\text{in}}(X) \subset C^{\text{in}}(Y)$ and $C^{\text{ex}}(X) = C^{\text{ex}}(Y)$.*

Proof. Since e is an inner edge $S_N(X) = S_N(Y)$, thus there is no edge $f \in C^{\text{in}}(X) \setminus C^{\text{in}}(Y)$. Since $e \notin C^{\text{in}}(X)$ and no edge in $C^{\text{ex}}(Y)$ is in $C^{\text{in}}(X)$, $C^{\text{in}}(X) \subset C^{\text{in}}(Y)$. Moreover, there is no cycle including $f \in C^{\text{ex}}(Y)$ in $G[S(X) \cup \{f\}]$, hence $C^{\text{ex}}(X) = C^{\text{ex}}(Y)$. □

Next, for any pair of edges e and f not in $G[S(Y)]$, we consider the computation of the girth of $G[S(Y) \cup \{e, f\}]$ in `EUG-ES`. Let $A(Y) = \{v \in S_N(Y) \mid E(v) \cap C(Y) \neq \emptyset\}$. In a similar fashion as `EUG-IS`, `EUG-ES` uses $M_Y^{\mathtt{ue}}$ for $A(Y)$. The definition of $M_Y^{\mathtt{ue}}$ is as follows: For any pair of vertices u and v in $A(Y)$, $M_Y^{\mathtt{ue}}(u, v)$ is the distance between u and v in $A(Y)$. Note that a shortest path between u and v may contain a vertex in $G[S(Y)] \setminus A(Y)$. The next lemma shows that by using $M_Y^{\mathtt{ue}}$, we can compute $C(X)$ in $O(|S_N(X)|)$ time from $C(Y)$.

Lemma 36. *For any iteration X, $\left|C^{\text{in}}(X)\right| \leq |S_N(X)|$.*

Proof. The proof follows from these facts: (A) Initially, $C^{\text{in}}(X) = \emptyset$. (B) Choosing $e \in C^{\text{in}}(Y)$ decreases $|C^{\text{in}}(X)|$, where Y is the parent iteration of X. (C) $e = \{x, y\} \in C^{\text{ex}}(Y)$ is chosen iff $C^{\text{in}}(Y) = \emptyset$, and (assuming wlog $y \notin S_N(Y)$) it increases $|C^{\text{in}}(X)|$ by at most $|\{\{y, z\} \mid z \in S_N(Y)\}| < |S_N(Y)|$. □

Lemma 37. *For any iteration X, $|C^{\text{ex}}(Y) \setminus C^{\text{ex}}(X)| + |C^{\text{ex}}(X) \setminus C^{\text{ex}}(Y)| \leq |S_N(X)|$, where Y is the parent iteration of Y.*

Algorithm 6: Updating data structures in `EUG-ES`.

```
1  Procedure NextCand(C(S), M_S^ue, S, g, G)
2      if C^in ≠ ∅ then  e ← C^in; else e ← C^ex ;
3      C(S ∪ {e}) ← UpdateCand(e, S);
4      M^ue_{S∪{e}} ← UpdateUE(e, C(S ∪ {e}));
5  Function UpdateCand(e = {u, v}, S)
6      if e ∈ C^in then
7          for f ∈ C^in \ {e} do
8              if g(G[S ∪ {e, f}]) ≥ g then  C^in ← C^in ∪ {f} ;
9      else                                   // We assume u ∈ S_N and v ∉ S_N
10         for w ∈ N(v) do                    // Let f be an edge {v, w}
11             if g(G[S ∪ {e, f}]) < g then  C^ex ← C^ex \ f ;
12             else if w ∈ S_N then  (C^in, C^ex) ← (C^in ∪ f, C^ex \ f) ;
13             else  C^ex ← C^ex ∪ f ;
14     return C^in ∪ C^ex;
15 Function UpdateUE(e = {u, v}, C(S ∪ {e}))
16     A = {v ∈ S_N | v is incident to C(S).};
17     for x, y ∈ A do                        // If e ∈ C^ex, then u ∈ S_N, v ∉ S_N
18         if e ∈ C^in then
19             M_S^ue(x, y) ←
                 min{M_S^ue(x, y), M_S^ue(x, u) + M_S^ue(v, y) + 1, M_S^ue(x, v) + M_S^ue(u, y) + 1};
20         else  M_S^ue(x, y) ← min{M_S^ue(x, y), M_S^ue(x, u) + 1} ;
21     return M_S^ue;
```

Proof. We consider two cases: (I) $C^{\text{in}}(Y) \neq \emptyset$: `EUG-ES` picks $e \in C^{\text{in}}(Y)$, and thus, from Lemma 35, $C^{\text{ex}}(X) = C^{\text{ex}}(Y)$. (II) $C^{\text{in}}(I) = \emptyset$: `EUG-ES` picks $e = \{u, v\} \in C^{\text{ex}}(Y)$. Without loss of generality, we can assume that $u \in S_N(Y)$ and $v \notin S_N(Y)$. Let f be an edge $\{v, w\}$ incident to v. Now, $w \in S_N(X)$. This implies that the number of edges that are added to $C^{\text{ex}}(X)$ and removed from $C^{\text{ex}}(Y)$ is at most $|S_N(X)|$. □

Note that $|S_N(Y)| \leq |S_N(X)|$. Hence, from the above lemmas, we can obtain the following lemma.

Lemma 38. *$C(X)$ can be computed in $O(|S_N(Y)|)$ time from $C(Y)$.*

Theorem 39. `EUG-ES` *enumerates all connected subgraphs with girth g in $O\left(\sum_{S \in \mathcal{S}} |S_N|\right)$ total time using $O\left(n \cdot \max_{S \in \mathcal{S}}\{|S_N|^2\}\right)$ space. Moreover, the delay is $O\left(\max_{S \in \mathcal{S}}\{|S_N|\}\right)$.*

Proof. The correctness of `EUG-IS` can be seen similarly to that of `EDG-ES`, i.e., Algorithm 2 adapted to the enumeration of edge subgraphs. Let $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ be the enumeration tree made by `EUG-ES`. We first consider the space complexity of `EUG-ES`. In each iteration X, `EUG-ES` needs $O\left(\max_{X \in \mathcal{V}}\{|A(X)|^2\}\right)$ for storing $M_X^{\mathtt{ue}}$. In addition, each cell of $M_X^{\mathtt{ue}}$ is updated at most n times since the distance is at most n and monotonically decreasing. `EUG-ES` traverses on $\mathcal{T}$ in a DFS manner. Hence, the space complexity of `EUG-ES` is $O\left(n \cdot \max_{S \in \mathcal{S}}\{|S_N|\}^2\right)$ since $A(X) \subseteq S_N(X)$ for any iteration $X \in \mathcal{V}$.

Let us consider the time complexity. Let Y be a parent of X. Suppose that we add $e = \{u, v\}$ to $S(Y)$, and $S(X) = S(Y) \cup \{e\}$, that is, Y is the parent iteration of X. Then, $M_X^{\mathtt{ue}}(x, y) = \min\{M_Y^{\mathtt{ue}}(x, y), M_{Y)}^{\mathtt{ue}}(x, u) + M_Y^{\mathtt{ue}}(v, y) + 1, M_Y^{\mathtt{ue}}(x, v) + M_Y^{\mathtt{ue}}(u, y) + 1\}$. Thus, we can compute $M_X^{\mathtt{ue}}$ from $M_Y^{\mathtt{ue}}$ in $O\left(|A(X)|^2\right)$ time since each value of $M_X^{\mathtt{ue}}$ can be computed in constant time. From Lemma 38, `EUG-ES` needs $O\left(|S_N(Y)| + |A(X)|^2\right)$ time for generating data structures for $S(X)$ from those for $S(Y)$. Thus, since $|S_N(Y)| \leq |S_N(X)|$, the total time of `EUG-ES` is $O\left(\sum_{X \in \mathcal{V}} |S_N(X)| + |A(X)|^2\right)$.

Note that, X has $|C(X)|$ child iterations. Moreover, $|A(X)|$ is at most $2|C(X)|$ since each vertex in $A(X)$ is incident to at least one edge in $C(X)$. Hence, $O\left(|A(X)|^2\right) = O\left(|C(X)| \cdot |A(X)|\right)$. Since $|\mathcal{V}| = 1 + \sum_{X \in \mathcal{V}} |C(X)|$, by delivering $O(A(X))$ time to each child of X, the time complexity of `EUG-ES` is $O\left(\sum_{X \in \mathcal{V}}(|S_N(X)| + |A(X)|)\right)$. In addition, $|A(X)|$ is at most $|S_N(X)|$ since $A(X) \subseteq S_N(X)$. Hence, this algorithm runs in amortized $O\left(\max_{S \in \mathcal{S}}\{|S_N|\}\right)$ time. □

Analysis of delay

While the cost per solution of all algorithms are $O(n)$. However, their *delay*, i.e., the maximum elapsed time between the output of a solution and the following one, is higher, unless we employ additional techniques. By outputting the solution at the beginning of every iteration, a solution will be output whenever an iteration is performed. In this case the delay will be bounded by $O(n^2)$ delay. However, we can reduce the delay by using the *output queue* and *alternative output* techniques [71]: Let X be an arbitrary iteration, T^* be an upper bound on the cost of X, and $\bar{T}$ an upper bound for the ratio

$$\text{(cost of processing the subtree of } X\text{) / (solutions found in the subtree)} \tag{4.3}$$

To reduce the delay, we will need to use a buffer which stores $\lceil 2 \cdot T^*/\bar{T} \rceil + 1$ solutions. First, we fill the buffer until it is full, then we will out a solution every $O(\bar{T})$ time, obtaining $O(\bar{T})$ delay.

By means of our amortized cost analysis, we have that $\bar{T}$ corresponds exactly to the cost per solution, that is $O(n)$ for all algorithms.

Thus we will have $T^* = O(n^2)$ and $\bar{T} = O(n)$, meaning that we will obtain delay $O(n)$, at the cost of storing $\Theta(n)$ solutions. As a solution of `EDG-IS` and `EUG-IS` are

defined by a set of vertices, this translates to a space usage of $O(n^2)$ and a delay of $O(n)$.

As for `EDG-ES` and `EUG-ES`, we need to store solutions corresponding to sets of edges, which have size $O(m)$ and take $O(m)$ to output. We address this problem with the alternative output technique. This consists in performing the output of a solution as the *first* operation in each iteration of *even* depth, and as the *last* operation in each iteration of *odd* depth.

By using this structure, consecutive outputs of the algorithm are performed by iterations at distance at most 3 in the enumeration tree (see Figure 3 in [71]). As each iteration outputs a solution that differs by 1 edge from those output by its parent and children, consecutively output solutions will differ by at most 3 edges. We can thus output each solution by giving only the difference with the last output solution, which takes constant space (and time), thus the buffer size will take only $O(m)$ space for the first solution, and $O(n)$ space for the subsequent n ones, for a total cost of $O(n)$ delay and $O(m)$ space usage.

In both cases, the space required by the solution buffer does not increase the $O(n^2)$ space usage of all algorithms. However, the output queue technique will add a pre-processing time, that is the time required to fill the buffer: as without the output queue technique the algorithm guarantee a delay of $O(n^2)$ time, the time required to fill a buffer of $\Theta(n)$ solution, that is $O(n^3)$.

Finally, we summary our result in Table 4.1. If g is equal to $n + 1$, the algorithms will enumerate all acyclic subgraphs. Furthermore, they can easily be adapted to relax the connectivity constraint and consider weighted graphs.

input graph	subgraph type	algorithm	delay	space usage
directed	induced	EDG-IS	$O(\max_{S\in\mathcal{S}}\{\lvert N[S]\rvert\})^*$	$O(n\cdot\max_{S\in\mathcal{S}}\{\lvert N[S]\rvert^2\})^\dagger$
	edge	EDG-ES	$O(\max_{S\in\mathcal{S}}\{\lvert S_N\rvert\})^*$	$O(n\cdot\max_{S\in\mathcal{S}}\{\lvert S_N\rvert^2\})^\dagger$
undirected	induced	EUG-IS	$O(\max_{S\in\mathcal{S}}\{\lvert N[S]\rvert\})^*$	$O(n\cdot\max_{S\in\mathcal{S}}\{\lvert N[S]\rvert^2\})$
	edge	EUG-ES	$O(\max_{S\in\mathcal{S}}\{\lvert S_N\rvert\})^*$	$O(n\cdot\max_{S\in\mathcal{S}}\{\lvert S_N\rvert^2\})$

Table 4.1: Summary of the time and space complexity of the proposed enumeration algorithms for enumerating subgraphs with girth at least g, where $\mathcal{S}$ is the set of solutions and S_N is the set of vertices incident to S. *: $O(n^3)$ pre-processing time is also required. †: the bound drops to $O(n^2)$ for enumerating acyclic subgraphs (i.e., $g = n + 1$).

Experimental result

We conducted experiments on artificial data to evaluate the computational speed. We implemented `EUG-ES`. The experimental environment is as follows: C++ with GCC7.3.0 and -O3 option. We considered general graphs(`GG`) randomly generated. We experimented three algorithms, `EUG-ES`, a simple polynomial delay algorithm, and a naive algorithm for `GG`. The time complexity of the simple algorithm and the naive algorithm are amortized $O(nm^2)$ time and $O(2^m poly(n))$ time. The efficient algorithm is 60 times faster than the simple polynomial delay algorithm.

#Edges	$O(n)$ and $O(nm^2)$	naive
23	380,906	$2^{24} - 1 = 8,388,607$
25	997,093	$2^{26} - 1 = 67,108,863$

Table 4.2: The number of iterations in connected bounded girth enumeration. An input graph is in `GG`.

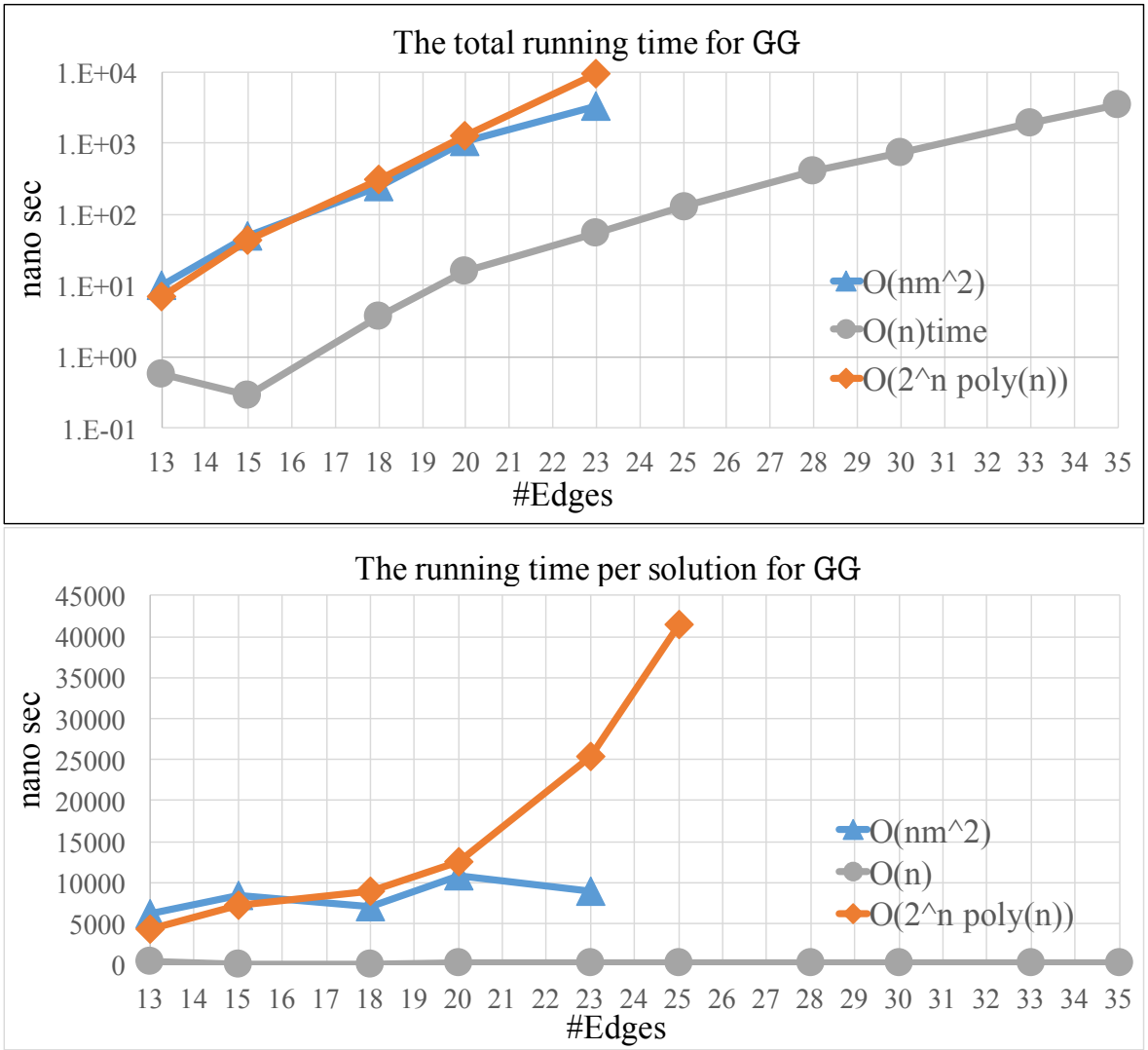


Figure 4.4: The total running time and the time per solution of connected bounded subgraph enumeration. All input graphs in GG have 10 vertices.

Chapter 5

Enumeration of Dominating Sets

5.1 Introduction

In this chapter, we address the dominating set enumeration problem. A set of vertex 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. A dominating set is one of a fundamental substructure of graphs. Finding the minimum dominating set problem is a classical NP-hard problem [25]. A dominating set is fixed parameter tractable if an input is sparse graphs [57, 59]. We show that our proposed algorithms solve this problem in constant amortized time if graphs are sparse.

The enumeration of *minimal* dominating sets of a graph is closely related to the enumeration of *minimal hypergraph transversals* [19]. A dominating set D is a *minimal* if $D \setminus \{v\}$ is not dominating set for any $v \in D$. Kanté *et al.* [35] 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 [18, 28, 34–37]. Incremental polynomial time algorithms have also been developed [5, 27, 29]. However, it is still open whether there exists an output polynomial time algorithm for enumerating minimal dominating sets from general graphs. We now define the dominating set enumeration problem as follows:

Problem 6 (The dominating set enumeration problem). *Given a graph G, then output all dominating sets in G without duplication.*

Main results

In this chapter, 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 53). Moreover, `EDS-G` enumerates all dominating sets in constant time per solution for a graph with girth at least nine (Theorem 66).

5.2 A basic algorithm based on reverse search

In this chapter, we propose two algorithms `EDS-D` and `EDS-G` for solving Problem 6. 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* that is proposed by Avis and Fukuda [3] and is one of the framework for constructing enumeration algorithms.

An algorithm based on reverse search enumerates solutions by traversing on an implicit tree structure on the set of solution, called a *family tree*. For building the family tree, 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 7. 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 40, 41, and 42, we can obtain the correctness of `EDS`.

Lemma 40. *For any dominating set X, by recursively applying the parent function $\mathcal{P}\,(\cdot)$ to X at most n times, we obtain V.*

Proof. For any dominating set X, since $pv\,(v)$ always exists, there always exists the parent vertex for X. In addition, $|\mathcal{P}\,(X) \setminus X| = 1$. Hence, the statement holds. □

Lemma 41. *$\mathcal{F}\,(G)$ forms a tree.*

Algorithm 7: `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: a 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);
```

Proof. Let X be any solution in $\mathcal{S}(G) \setminus \{V\}$. Since X has exactly one parent and V has no parent, $\mathcal{F}(G)$ has $|V(\mathcal{F}(G))| - 1$ edges. In addition, since there is a path between X and V by Lemma 40, $\mathcal{F}(G)$ is connected. Hence, the statement holds. □

Lemma 42. *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 $X = Y \cup \{v\}$.*

Proof. The if part is immediately shown from the definition of a candidate set. We show the only if part by contradiction. Let Z be a dominating set in $ch(X)$ such that $Z = X \setminus \{v'\}$, where $v' \in Z$. We assume that $v' \notin C(X)$. From $v' \notin C(X)$, $N[\mathcal{P}(Z)] \neq V$ or $\mathcal{P}(Z) \neq X$. Since Z is a child of X, $\mathcal{P}(Z) = X$, and thus, $N[\mathcal{P}(Z)] = V$. This contradicts $v' \notin C(X)$. Hence, the statement holds. □

Theorem 43. *By traversing $\mathcal{F}(G)$,* `EDS` *solves Problem 6.*

5.3 The algorithm 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

Algorithm 8: `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} ;
```

$O(\Delta^2)$ 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* [48] 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 Figure 5.1. In Figure 5.1, each vertex v is adjacent to vertices at most two whose

Algorithm 9: Several ancillary functions to compute `EDS-D`.

```
1  Procedure Cand-D(X, v, C)
2      Y ← X \ {v}; Del_1 ← ∅; Del_2 ← ∅;
3      for u ∈ (N(v) ∩ C) ∪ N(v)_{v<} do
4          if u < v then
5              if N(u)_{u<} ∩ Y = ∅ ∧ N(u)_{<u} ∩ Y = ∅ then  Del_1 ← Del_1 ∪ {u} ;
6          else
7              if N[u] ∩ (X \ C) = ∅ ∧ |N[u] ∩ C| = 2 then
                   Del_2 ← Del_2 ∪ (N[u] ∩ C) ;
8      return C \ (Del_1 ∪ Del_2);                          // C is C(X \ {v})
9  Procedure DomList (v, Y, X, C' ⊕ C(Y), D')
10     for u ∈ C' ⊕ C(Y) do
11         for w ∈ N(u)_{u<} do
12             if u ∉ D'_w(X) then
13                 if u ∉ C' then  D'_w ← D'_w ∪ {u} ;
14                 else  D'_w ← D'_w \ {u} ;
15     for u ∈ N(v)_{v<} do
16         if u ∈ X then  D'_v ← D'_v ∪ {u} ;
17     return D';                                           // D' is D(Y)
```

indices are larger than v. Matula and Beck show that the degeneracy and a degeneracy ordering of G can be obtained in $O(n+m)$ time [53]. Our proposed algorithm `EDS-D`, shown in Algorithm 8, 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

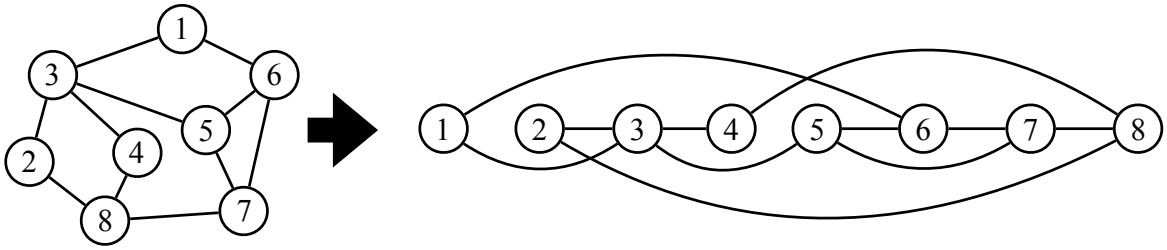


Figure 5.1: An example of a degeneracy ordering for a 2-degenerate graph G.

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

Proof. Let Z be a child of Y. Hence, $pv(Z) \in X$ and $pv(Z) \in C(Y)$. From the definition of $pv(Z)$, $pv(Z) = \min V \setminus Z$. Moreover, since $V \setminus X \subset V \setminus Z$, $pv(Z) \leq \min V \setminus X$. Therefore, $pv(Z) \in C(X)$. □

From the Lemma 44, 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 $Del(X, pv(Y))$.

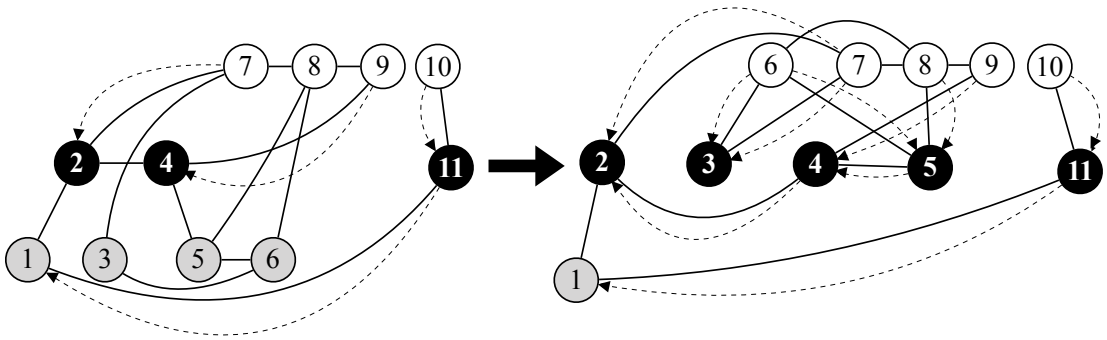


Figure 5.2: Let X be a dominating set $\{1, 2, 3, 4, 5, 6, 11\}$. An example of the maintenance of $C(X)$ and $\mathcal{D}(X)$. Each dashed directed edge is stored in $\mathcal{D}(X)$, and each solid edge is an edge in G. A directed edge (u, v) implies $v \in D_u(X)$. White, black, and gray vertices belong to $V \setminus X$, $X \setminus C(X)$, and $C(X)$, respectively.

Lemma 45. *For each $v \in C(X)$, $Del(X, v) = Del_1(X, v) \cup Del_2(X, v) \cup Del_3(X, v)$ holds.*

Proof. $Del(X, v) \supseteq Del_1(X, v) \cup Del_2(X, v) \cup Del_3(X, v)$ is trivial since $X \setminus \{u, v\}$ is not dominating set for each $u \in Del_1(X, v) \cup Del_2(X, v)$ and the parent of $X \setminus \{u, v\}$ is not $X \setminus \{v\}$ for each $u \in Del_3(X, v)$. We next prove $Del(X, v) \subseteq Del_1(X, v) \cup Del_2(X, v) \cup Del_3(X, v)$. Let u be a vertex in $Del(X, v)$. Suppose that $X \setminus \{u, v\}$ is a dominating set. Since $\mathcal{P}(X \setminus \{u, v\}) \neq X \setminus \{v\}$, $v < u$. Thus, $u \in Del_3(X, v)$. Suppose that $X \setminus \{u, v\}$ is not a dominating set, that is, $N[X \setminus \{u, v\}] \neq V$. This implies that there exists a vertex w in V such that w is not dominated by any vertex in $X \setminus \{u, v\}$. Note that w may be equal to u or v. Hence, $N[w] \cap X = \{u, v\}$ and the statement holds. □

We show an example of dominated list and a maintenance of $C(X)$ in Figure 5.2. 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 46. *For each vertex $v \in C(X)$, we can compute $N(v) \cap C(X)$ and $N(v)_{v<} \cap X$ in $O(k)$ time on average over all children of X.*

Proof. Since G is k-degenerate, $G[C(X)]$ is also k-degenerate. Thus, the number of edges in $G[C(X)]$ is at most $k|C(X)|$. Remind that $C(X)$ is sorted by the degeneracy ordering. Hence, by scanning vertices of $C(X)$ from the smallest vertex to the largest one, for each v in $C(X)$, we can obtain $N(v) \cap C(X)$ in $O(k)$ time on average over all children of X. Since $N(v)_{v<}$ is the larger v's neighbors set, the size is at most k. Hence, the statement holds. □

Lemma 47. *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.*

Proof. Since every vertex u in $Del_1(X, v)$ is adjacent to v, $Del_1(X, v) \subseteq N(v) \cap C(X)$. To compute $Del_1(X, v)$, we need to check whether $N[u] \cap X = \{u, v\}$ or not. We first consider smaller neighbors of u. Before computing $Del_1(X, v)$ for every vertex v, we record the size of D_u of $u \in C(X)$ in $O(|C(X)|)$ time. $D_u = \emptyset$ if and only if there are no smaller neighbors of u in $X_{<u} \setminus C(X)$. Moreover, the number of edges in $G[C(X)]$ is at most $k|C(X)|$ from the definition of the degeneracy. Thus, this part can be done in $O\left(\sum_{v \in C(X)} |N(v) \cap C(X)|\right)$ total time and in $O(k)$ time per each vertex in $C(X)$. We next consider larger neighbors. Again, before computing $Del_1(X, v)$ for

every vertex v, from Lemma 46 and the degeneracy of G, we can check all of the larger neighbors of $u \in C(X)$ in $O(k|C(X)|)$ time. Thus, as with the smaller case, the checking for the larger part also can be done in $O(k)$ time on average over all children of X. Hence, the statement holds. □

Lemma 48. *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.*

Proof. Let u be a vertex in $Del_2(X, v)$. Then, there exists a vertex w such that $N[w] \cap X = \{u, v\}$ and $w \in N[v] \cap (V \setminus (X \setminus \{v\})$. In addition, for any vertex v' in $C(X)$, $pv(X \setminus \{v'\}) = v'$. Thus, $v \leq w$ and $u < w$ hold. Before computing $Del_2(X, v)$ for every vertex v, by scanning all larger neighbors w' of vertices of $C(X)$, we can list such vertices w' such that $w' > \max\{C(X)\}$, $|N[w'] \cap C(X)| = 2$, and $w' \in V \setminus (X \setminus \{v\})$ in $O(k|C(X)|)$ time since G is k-degenerate. If $D_{w'} \neq \emptyset$, that is, w' has a neighbor in $X \setminus C(X)$, then $|N[w] \cap X| > 2$. Thus, since we can check the size of $D_{w'}$ in constant time, we can compute $Del_2(X, v)$ in $O(k)$ time on average over all children of X. □

In Lemma 47 and Lemma 48, 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 49. *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 $u \neq v$, $D_u(Y) = D_u(X) \cup (N(u)_{<u} \cap (Del_1(X, v) \cup Del_2(X, v))) \cup (N(u)_{<u} \cap (Del_3(X, v) \setminus \{v\}))$.*

Proof. Let $X_{\bar{C}} = X \setminus C(X)$. Suppose that $u \in Y$. From the definition, $D_u(X) = N(u)_{<u} \cap X_{\bar{C}}$. From the distributive property,

$$\begin{aligned} L &= D_u(X) \cup (N(u)_{<u} \cap (Del_1(X,v) \cup Del_2(X,v))) \cup (N(u)_{<u} \cap (Del_3(X,v) \setminus \{v\})) \\ &= N(u)_{<u} \cap (X_{\bar{C}} \cup (Del(X,v) \setminus \{v\})) \\ &= N(u)_{<u} \cap (Y \setminus C(Y)) \end{aligned}$$

Since $X_{\bar{C}} \cup (Del(X,v) \setminus \{v\}) = Y \setminus C(Y)$. Suppose that $u \in V \setminus X$. From the parent-child relation, $pv(Y) < u$ holds. Since $Del(X,v) \subseteq V_{<u}$, $N(u)_{<u} \cap (Del_1(X,v) \cup Del_2(X,v)) = N(u) \cap (Del_1(X,v) \cup Del_2(X,v))$, and $N(u)_{<u} \cap (Del_3(X,v) \setminus \{v\}) = N(u) \cap (Del_3(X,v) \setminus \{v\})$. From the definition, $D_u(X) = N(u) \cap X_{\bar{C}}$,

$$\begin{aligned} L &= D_u(X) \cup (N(u)_{<u} \cap (Del_1(X,v) \cup Del_2(X,v))) \cup (N(u)_{<u} \cap (Del_3(X,v) \setminus \{v\})) \\ &= (N(u) \cap X_{\bar{C}}) \cup (N(u) \cap (Del_1(X,v) \cup Del_2(X,v))) \cup (N(u) \cap (Del_3(X,v) \setminus \{v\})) \\ &= N(u) \cap (X_{\bar{C}} \cup (Del_1(X,v) \cup Del_2(X,v)) \cup (Del_3(X,v) \setminus \{v\})) \\ &= N(u) \cap (X_{\bar{C}} \cup (Del(X,v) \setminus \{v\})) \\ &= N(u) \cap (Y \setminus C(Y)) \end{aligned}$$

Hence, the statement holds. □

Lemma 50. *Let X be a dominating set, v be a vertex in $C(X)$, and $Y = X \setminus \{v\}$. $D_v(Y) = D_v(X) \cup (N(v)_{<v} \cap (Del_1(X,v) \cup Del_2(X,v))) \cup (N(v)_{v<} \cap X)$.*

Proof. Since $Del_1(X,v) \cup Del_2(X,v) \subseteq V_{<v}$ and $Del_3(X,v) \cap V_{<v} = \emptyset$, $N(v)_{<v} \cap (Del_1(X,v) \cup Del_2(X,v)) = N(v)_{<v} \cap Del(X,v)$. By the same discussion as Lemma 49, $L = D_v(X) \cup (N(v)_{<v} \cap Del(X,v)) = N(v)_{<v} \cap (Y \setminus C(Y))$. Since $Y = X \setminus \{v\}$, $N(v)_{v<} \cap Y = N(v)_{v<} \cap X$. Moreover, since $X_{<v} = Y_{<v}$ and $C(Y)_{v<} = \emptyset$, $N(v)_{v<} \cap (Y \setminus C(Y)) = N(v)_{v<} \cap X$. Since $L = (N(v)_{v<} \cup N(v)_{<v}) \cap (Y \setminus C(Y)) = D_v(Y)$, the statement holds. □

We next consider the time complexity for obtaining the dominated lists for children of X. From Lemma 49 and Lemma 50, 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 51. *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. $\square$

Lemma 52. *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 47 and Lemma 48, `Cand-D` correctly computes $Del_1(X, v)$ and $Del_2(X, v)$ in from line 4 to line 5 and from line 6 to line 7, 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 51, 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 53. `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 44 and Lemma 45, `EDS-D` correctly computes all children. Hence, the correctness of `EDS-D` is

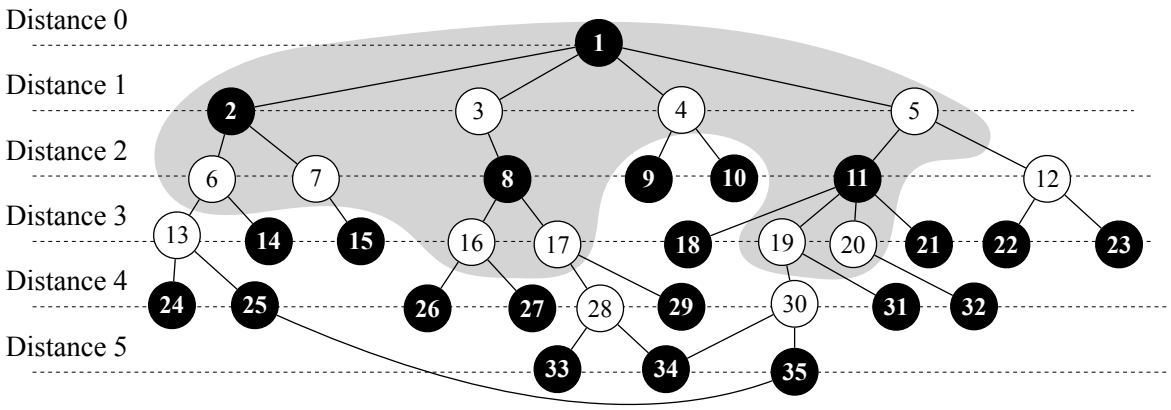


Figure 5.3: An example of $G_v(X)$, where $v = 1$. The vertices in the grey area are $Del'(X, v) \cup (G_v(X) \setminus G(Y)) \cup (N'_v(v) \setminus X)$. Each horizontal line represents the distance between 1 and any vertex.

shown by the same manner of Theorem 43. 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 52. 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. $\square$

Algorithm 10: `EDS-G` enumerates all dominating sets in $O(1)$ time per solution for a graph with girth at least nine.

```
Procedure EDS-G(G = (V, E))                          // G: an input graph
    for v ∈ V do  f_v ← False ;
    AllChildren (V, V, {f_1, ..., f_|V|}, G);
Procedure AllChildren (X, C, F, G)
    Output X;
    for v ∈ C(X) do                          // v is the largest vertex in C
        Y ← X \ {v};
        (C(Y), F(Y), G(Y)) ← Cand-G (v, C, F, G);
        AllChildren (Y, C(Y), F(Y), G(Y));
        for u ∈ N_G(v) do
            if u ∈ C then  f_u ← True ;
            else  G ← G \ {u} ;
        G ← G \ {v};
        C ← C \ {v};                    // Remove vertices in Del_3(X, v).
```

5.4 The algorithm for graphs with girth at least nine

In this section, we propose an efficient 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 10. To achieve constant amortized time enumeration, we focus on the *local structure* $G_v(X)$ for (X, v) of G defined as

Algorithm 11: An ancillary function to compute `EDS-G`.

```
1  Procedure Cand-G (v, C, F, G)
2      Del_1 ← ∅; Del_2 ← ∅;
3      for u ∈ N_G(v) do
4          if N_G[u] ∩ X = {u, v} and f_u = False then  Del_1 ← Del_1 ∪ {u} ;
5          else if ∃w(N_G[u] ∩ X = {w, v}) then  Del_2 ← Del_2 ∪ {w} ;
6      C' ← C \ (Del_1 ∪ Del_2 ∪ {v});
7      for u ∈ N'[Del_1 ∪ Del_2] do                                  // Lemma 57
8          f_u ← True;
9          if u ∉ C' then  G ← G \ {u} ;
10     if f_v = True then G ← G \ {v};
11     return (C', F, G);
```

follows: $G_v(X) = G[(V \setminus N[X \setminus C(X)_{\leq v}]) \cup C(X)_{\leq v}]$. Figure 5.3 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 45, $C(X) \setminus C(Y) = Del(X, v)$, where $X = Y \cup \{v\}$. We denote by $f_v(u, X) = \texttt{True}$ if there exists a

neighbor w of u such that $w \in X \setminus C(X)_{\leq v}$; Otherwise $f_v(u, X) = \texttt{False}$. Thus, `Cand-G` correctly computes $Del_1(X, v)$ and $Del_2(X, v)$ from line 3 to 5. 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 43, 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.

Lemma 54. *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 $f_v(u, X) = \texttt{False}$.*

Proof. The only if part is obvious since $u, v \in C(X)_{\leq v}$ and $N[u] \cap X = \{u, v\}$. We next prove the if part. Since $f_v(u, x) = \texttt{False}$, $N[u] \cap (X \setminus C(X)_{\leq v}) = \emptyset$. Moreover, since $(N'_v[u] \cap X) \subseteq C(X)_{\leq v}$, $N[u] \cap X = N'_v[u] \cup (N[u] \cap (X \setminus C(X)_{v<})) = \{u, v\}$. Hence, the statement holds. □

Lemma 55. *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 $N'_v[w] \cap X = \{u, v\}$.*

Proof. The only if part is obvious since $u, v \in C(X)_{\leq v}$ and there is a vertex w such that $N[w] \cap X = \{u, v\}$. We next show the if part. Since $w \in G_v(X)$, $w \in C(X)_{\leq v}$ or $w \notin X \cup N[X \setminus C(X)_{\leq v}]$. Moreover, since $N'[w] = \{u, v\}$, $w \notin X$, that is, $w \notin C(X)$. Hence, $w \notin N[X \setminus C(X)_{\leq v}]$. Therefore, $N[w] \cap X = (N'_v[w] \cap X) \cup (N[w] \cap (X \setminus C(X)_{v<})) = \{u, v\}$ and the statement holds. □

Lemma 56. *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*

Proof. Since $Del_3(X, v) \cap C(X \setminus \{v\}) = \emptyset$, $C(X \setminus \{v\}) \subseteq C(X)_{\leq v}$. Thus, we do not need to remove vertices in $Del_3(X, v)$ from $C(X)_{\leq v}$. From Lemma 54, for each vertex $u \in N'_v(v)$, we can check whether $u \in Del_1(X, v)$ or not in constant time by confirming that $f_v(u, X) = \texttt{False}$ and $|N'_v(u)| = 2$. Moreover, from Lemma 55, for each vertex $w \in N'_v(v)$, we can compute $Del_2(X, v)$ by listing vertices in $u \in C(X)_{\leq v}$ such that $N'[w] \cap X = \{u, v\}$ or not. Note that since any vertex in $X_{<v}$ belongs to X, $N'[w] \cap X = \{u, v\}$ if $f_v(w, X) = \texttt{False}$, $|N'[w]| = 2$, and u and v are adjacent to w. Hence, the statement holds. □

Lemma 57. *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 $d'_v(u) = |N'_v(u)|$.*

Proof. From the definition, $V(G(Y)) \subseteq V(G_v(X))$. Let us denote by u a vertex in $G_v(X)$ but not in $G(Y)$ such that $u \neq v$. This implies that (A) u is dominated by some vertex in $Y \setminus C(Y)$ and (B) $u \notin C(Y)$. Thus, for any vertex $u' \notin N'_v[Del'(X, v) \setminus \{v\}]$,

$u' \in G_v(X)$ if and only if $u' \in G(Y)$. Hence, we can find such vertex u by checking whether for each vertex $w \in N'_v[Del'(X,v)]$, w satisfies (A) and (B). Before checking, we first update the value of f. This can be done by checking all the vertices in $N'_v[Del'(X,v)]$ and in $O(1)$ time per vertex. Hence, this update needs $O\left(\sum_{w \in Del'(X,v)} d'_v(w)\right)$ time. If w satisfies these conditions, that is, $f_v(w,X) = \texttt{False}$, $f(w,Y) = \texttt{True}$, and (B), then we remove w and edges that are incident to w from $G_v(X)$. This needs $O\left(\sum_{w \in G_v(X) \setminus G(Y)} d'_v(w)\right)$ total time for removing vertices. Thus, the statement holds. □

From Lemma 56 and Lemma 57, 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, 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 58. *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:*

$$O\left(\sum_{u \in Del'(X,v)} d'_v(u) + \sum_{u \in G_v(X) \setminus G(Y)} d'_v(u) + \sum_{u \in N'_v(v) \setminus X} d'_v(u)\right). \tag{5.1}$$

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

Lemma 59. *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 $|C(X)|$.*

Proof. Let u be the largest vertex in $C(X)_{<v}$ and w be a vertex in $G_v(X) \cap Del'(X,v)$. If $w \in Del_1(X,v)$, then $d'_u(w) = 0$ since $w \in N'_v(v)$. Otherwise, $w \in Del_2(X,v)$, then

$d'_u(w) = 0$ since a vertex x such that $N'_v[x] = \{w, v\}$ is removed from $G_v(X)$. Hence, $Pen_v(X) \cap Pen_u(X) = \emptyset$. Moreover, for each $v \in C(X)$, $Pen_v(X)$ is a subset of $C(X)$. Hence, the union of $Pen_v(X)$ is a subset of $C(X)$ for each $v \in C(X)$. □

Let v be a vertex in $C(X)$ and a pendant in $G_v(X)$. A vertex v *pendant* if the degree of v is one. 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 59, 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 60. *Let X be a dominating set, v be a vertex in $C(X)$, and u be a vertex in $G_v(X)$. Then, $\left|N'_v[u] \cap C(X)_{\leq v}\right| \geq 2$ if $u \notin C(X)$. Otherwise, $\left|N'_v[u] \cap C(X)_{\leq v}\right| \geq 1$.*

Proof. If $u \in C(X)$, then $u \in N'[u] \cap C(X)$. We assume that $u \notin C(X)$. Thus, $N'[u] \cap (X \setminus C(X)) = \emptyset$ from the definition of $G(X)$. If $|N'[u] \cap C(X)| = 0$, then u is not dominated by any vertex. This contradicts X is dominating set. If $|N'[u] \cap C(X)| = 1$, then u is dominated only by the neighbor w of u in $C(X)$. This contradicts $w \in C(X)$. Hence, $|N[v] \cap C(X)| \geq 2$ if $v \notin C(X)$. □

Lemma 61. *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 $|(N'_v(v) \cap X) \setminus Del'(X, v)|$.*

Proof. Let u be a vertex in $(N'_v(v) \cap X) \setminus Del'(X, v)$. If $u \in C(X)$, then u is also a candidate vertex in $C(Y)$ since $u \notin Del'(X, v)$. Suppose that $u \notin C(X)$. Since $u \in G_v(X)$, u is dominated by only candidate vertices of X. However, since $u \in X$, u dominates it self and thus, this contradicts. Hence, the statement holds. □

Lemma 62. *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 $\sum_{u \in N'_v(v) \setminus X} (d'_v(u) - 1)$.*

Proof. Let u be a vertex in $N'_v(v) \setminus X$. That is, $u \notin C(X)$ and $N'_v(u) \subseteq C(X)$. Thus, from Lemma 60, there is a vertex $w \in N'_v(u)$ such that $w < v$. We consider the following two cases: (A) If $N'_v(u) = \{v, w\}$, then $w \in Del'(X, v)$. From the assumption, w has at least one neighbor x such that $x \neq u$. If $x \notin C(X)$, then there is a neighbor $y \in C(X)$ such that $y \neq w$. Suppose that $y \in Del'(X, v)$. This implies that there is a cycle with length at most six. This contradicts the girth of G. Hence, $y \notin Del'(X, v)$ and $Y \setminus \{y\}$ is a dominating set. If $x \in C(X)$, then $x \notin Del'(X, v)$ from the definition of $Del'(X, v)$ and the girth of G. Hence, $Y \setminus \{x\}$ is a dominating set. (B) Suppose $N'_v(u)$ has a vertex $z \in C(X)$ such that $z \neq v$ and $z \neq w$. If both z and w are in $Del'(X, v)$, then from the definition of $Del'(X, v)$ and the girth of G, G has a cycle with length at most five. Thus, without loss of generality, we can assume that $z \notin Del'(X, v)$. This allows us to generate a child $Y \setminus \{z\}$ of Y. Since the girth of G is at least nine, all children of Y generated above are mutually distinct. Hence, the statement holds. □

Lemma 63. *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 $\sum_{u \in Del'(X,v) \setminus \{v\}} (d'_v(u) - 1)$.*

Proof. Let u be a vertex in $Del'(X, v) \setminus \{v\}$. From the assumption, there is a neighbor w of u in $G(X)$. We consider the following two cases: (A) Suppose that w is in $G(Y)$. Since u is in $Y \setminus C(Y)$, $w \in C(Y)$. Hence, $Y \setminus \{w\}$ is a child of Y. Suppose that for any two distinct vertices x, y in $Del'(X, v) \setminus \{v\}$, they have a common neighbor w' in $G(Y)$. If both x and y are in $Del_2(X, v)$, then there exist two vertex z_x, z_y such that $N'_v[z_x] \cap X = \{x, v\}$ and $N'_v[z_y] \cap X = \{y, v\}$, respectively. Therefore, there is a cycle $(v, z_x, x, w', y, z_y, v)$ with length six. As with the above, if x or y in $Del_1(X, v)$, then there exists a cycle with length less than six since $\{x, v\} \in G$ or $\{x, v\} \in G$. This contradicts of the assumption of the girth of G. Hence, any pair vertices in $Del'(X, v)$ has no common neighbors. Thus, in this case, all grandchildren of X are

mutually distinct. (B) Suppose that w is not in $G(Y)$. Thus, if $w \in C(X)$, then $w \in Del'(X, v)$. This implies that there is a cycle including w and u whose length is less than six. Hence, w is not in $C(X)$. Then, from Lemma 60, there is a vertex z in $N'_v(w) \cap C(X)$ such that $z \neq u$. Since $u \in Del'(X, v) \setminus \{v\}$, there is an edge between u and v, or there is a vertex c such that $\{u, c\}$ and $\{v, c\}$ are in $G_v(X)$. Again, if z is in $Del'(X, v)$, then there is a cycle with length less than seven. Thus, z still belongs to $C(Y)$ and $X \setminus \{v, z\}$ is a dominating set. Next, we consider the uniqueness of $X \setminus \{v, z\}$. If there is a vertex w' such that $w' \in N'_v(u)$, $w' \neq w$, w and w' share a common neighbor u' other than u, then (u, w, u', w') is a cycle. Hence, any pair neighbors of u has no common neighbors. As with the above, any two distinct vertices in $Del'(X, v) \setminus \{v\}$ also has no common vertex like z. If there are two distinct vertex $u, u' \in Del'(X, v)$ such that u and u' has a common vertex like z, then there is a cycle with length at most eight even if $u, u' \in Del_2(X, v)$. This contradicts the assumption of the girth, and thus, the statement holds. □

Lemma 64. *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*

$$\sum_{u \in G_v(X) \setminus (G(Y) \cup Del'(X,v) \cup N'_v(v))} (d'_v(u) - 1).$$

Proof. Let u be a vertex in $G_v(X) \setminus (G(Y) \cup Del'(X, v) \cup N'_v(v))$. Since $u \notin Del'(X, v)$ and $u \in G_v(X) \setminus G(Y)$, u is not in X. Since $\left|N'_v(u) \cap C(X)_{\leq v}\right|$ is greater than or equal to two from Lemma 60, there are two distinct vertices w, w' in $N'_v(u)$. We assume that $w, w' \in Del'(X, v)$. From Lemma 45, the distance between w and v is at most two. Similarly, the distance between w' and v is at most two. Hence, there is a cycle with the length at most six since $w \neq v$ and $w' \neq v$. Thus, without loss of generality, we can assume that $w \notin Del'(X, v)$. (A) Suppose that $|N'_v(u)| = 2$.

If there is a vertex $u' \in G_v(X) \setminus (G(Y) \cup Del'(X,v) \cup N'_v(v))$ such that $u' \neq u$ and $w \in N'(u)$, then as with Lemma 62, there is a short cycle. Hence, for each vertex such as u, there is a corresponding dominating set $X \setminus \{v, w\}$. (B) Suppose that there is a neighbor $w'' \in N'_v(u) \cap C(X)$. Then, as mentioned in above, there is a dominating set $X \setminus \{v, w, w''\}$. In addition, by the same discussion as Lemma 63, such generated dominating sets are mutually distinct. (C) Suppose that there is a neighbor $w'' \in N'_v(u) \setminus C(X)$. From Lemma 60, there are two vertices $z, z' \in N'(w'') \cap C(X)$. Then, $z \notin Del'(X,v)$ or $z' \notin Del'(X,v)$, and thus, we can assume that $z \notin Del'(X,v)$. Therefore, there is a dominating set $X \setminus \{v, w, z\}$. Next, we consider the uniqueness of grandchildren of Y. Moreover, if there is a vertex u' such that $w, y \in N'(u')$ holds, such that $z \in N'(y)$. Then, there is a cycle (u, w, u', y, z, w'') with the length six. Hence, grandchildren of Y are mutually distinct for each $u \in G(X) \setminus G(Y) \setminus Del'(X,v)$. Thus, from (A), (B), and (C), the statement holds. □

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 61, Lemma 62, Lemma 63, and Lemma 64, we can obtain the following lemma:

Lemma 65. *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:*

$$\Omega\left(|C(X)| + \sum_{v \in C(X)} \left(\sum_{u \in Del'(X,v)} d'_v(u) + \sum_{u \in G_v(X) \setminus G(Y)} d'_v(u) + \sum_{u \in N'_v(v) \setminus X} d'_v(u) \right)\right). \tag{5.2}$$

From Lemma 58, Lemma 59, and Lemma 65, each iteration outputs a solution in constant amortized time. Hence, by the same discussion of Theorem 53, we can obtain the following theorem.

Theorem 66. *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 43, Lemma 54, and Lemma 55. By the same discussion with Theorem 53, the space complexity of `EDS-G` is $O(n+m)$. We next consider the time complexity of `EDS-G`. From Lemma 58, Lemma 59, and Lemma 65. 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. □

Experimental result

We conducted experiments on artificial data to evaluate the computational speed. We implemented `EDS-D`. The experimental environment is as follows: C++ with GCC7.3.0 and -O3 option. We considered 4-degenerate graphs(`DK`). We experimented three algorithms, `EDS-D`, a simple polynomial delay algorithm, and a naive algorithm for `DK`. The time complexity of the simple algorithm and the naive algorithm are amortized $O(nm)$ time and $O(2^n poly(n))$ time. The efficient algorithm is two times faster than the $O(nm)$ delay algorithm. However, this algorithm is slower than the $O(n\Delta)$ delay algorithm.

Interestingly, the fastest algorithm is a simple $O(n\Delta)$ time algorithm and a naive algorithm and `EDS-D` have the almost same running time. In Table 5.1, the number of iteration of the naive algorithm is three times of the number of iteration of `EDS-D`. It means that there is no redundant iteration in the naive algorithm. While, a constant factor of `EDS-D` in each iteration would be large. We conclude that the naive algorithm

and `EDS-D` have almost same running time for above reason.

#Nodes	$O(k)$	$O(n\Delta)$ and $O(nm)$	naive
25	26,616,233	54,947,204	$2^{26} - 1 = 67,108,863$
30	828,807,337	1,725,659,002	$2^{31} - 1 = 2,147,483,647$

Table 5.1: The number of iterations in dominating set enumeration. An input graph is in `DK`.

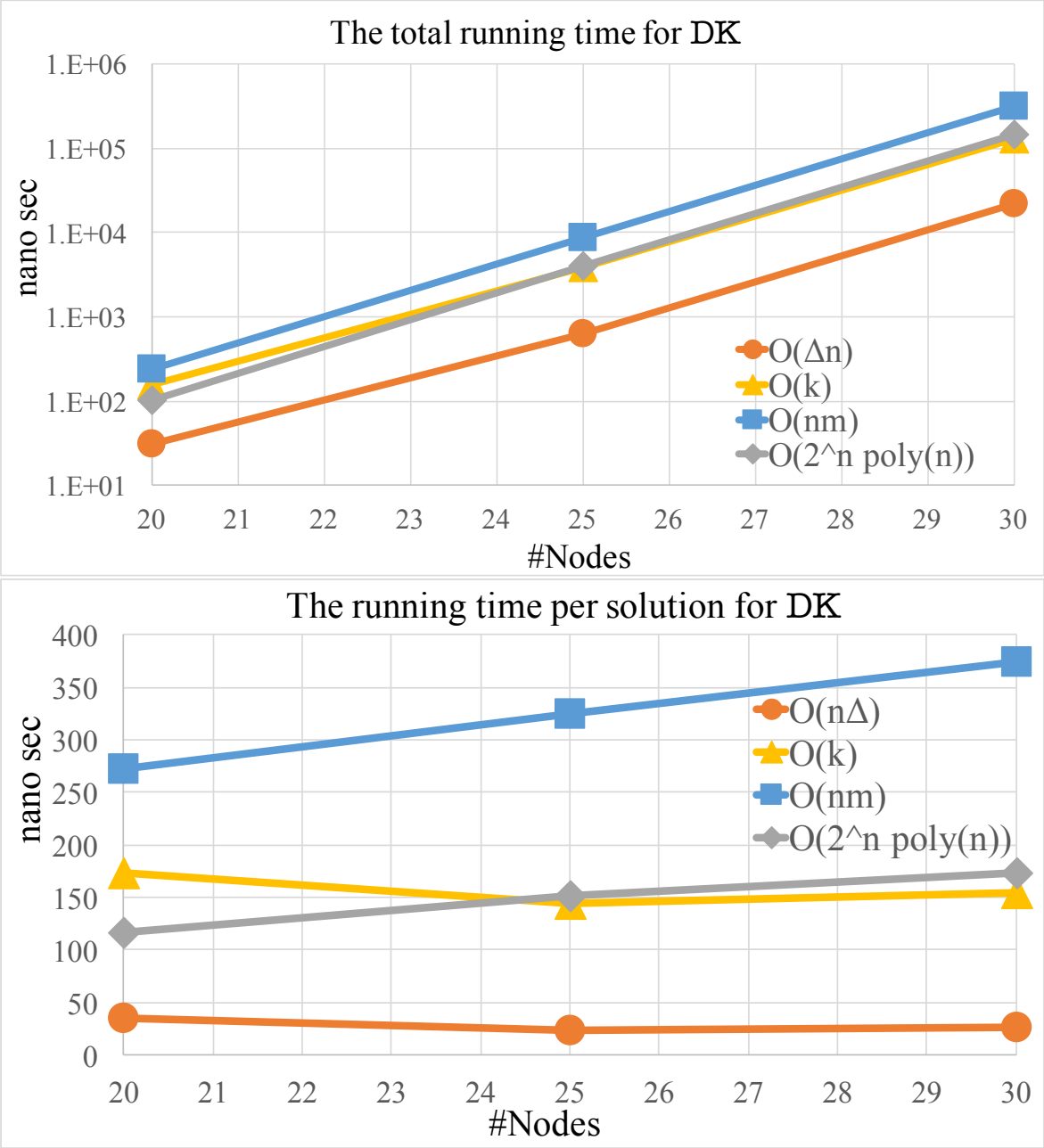


Figure 5.4: The total running time and the time per solution of dominating set enumeration. The degeneracy of a graph in DK is four.

Chapter 6

Enumeration of Chordal Bipartite Induced Subgraphs

6.1 Introduction

In this chapter, we consider the chordal bipartite induced subgraph enumeration. A *chordal bipartite graph* is a bipartite graph without induced cycles with length six or more. It is easy to solve recognition problems of a dominating set and an induced matching. By checking neighborhood of each element, we can recognize a dominating set and an induced matching. Hence, we can manage a candidate set in two problems in $O(poly(\Delta))$ time. However, it is difficult to recognize chordal bipartite graphs in $O(poly(\Delta))$ time. Thus, the management of a candidate set is difficult in $O(poly(\Delta))$ time.

We overcome this difficulty by defining the good parent child relation. For this reason, we introduce a new vertex elimination ordering, called a *chordal bipartite elimination ordering* (`CBEO`, in short). A vertex ordering `CBEO` is defined by the following

operation: Recursively remove a weak-simplicial vbertex [67]. A vertex ordering `CBEO` plays the key role in `ECB-IS`. In Figure 6.1, we show an input graph G and one of the solutions. We define our problem as follows.

Problem 7 (The chordal bipartite induced subgraph enumeration problem). *Output all chordal bipartite induced subgraphs in an input graph without duplication.*

Main result

Our proposed algorithm `ECB-IS` is based on the *reverse search* [3]. To construct an efficient enumeration algorithm, we have to give a *good* family tree, that is, the *good* parent for each solution. A vertex ordering `CBEO` plays the key role for defining such parents. As the main result of this chapter, we propose `ECB-IS` which outputs all chordal bipartite induced subgraphs in a given graph G in amortized $O(kt\Delta^2)$ time, where k is the degeneracy of G, t is the maximum size of $K_{t,t}$ as an induced subgraph, and Δ is the maximum degree of G. Moreover, the space usage of `ECB-IS` is $O(n+m)$.

6.2 A characterization of chordal bipartite graphs

We propose a new characterization of chordal bipartite graphs. By using this characterization, we construct our algorithm `ECB-IS` in Section 6.3. We first give some definitions. Let U be a subset of V. For vertices $u, v \in V$, u and v are *comparable* if $N(v) \subseteq N(u)$ or $N(v) \supseteq N(u)$ hold. Otherwise, u and v are *incomparable*. A vertex v is *weak-simplicial* [67] if $N(v)$ is an independent set and any pair of neighbors of v are comparable. A bipartite graph $B = (X, Y, E)$ is *bipartite chain* if any pair of vertices in X or Y are comparable, that is, $N(u) \subseteq N(v)$ or $N(u) \supseteq N(v)$ holds for any $u, v \in X$ or $u, v \in Y$. A bipartite graph B is *biclique* if any pair of vertices $x \in X$ and $y \in Y$ are

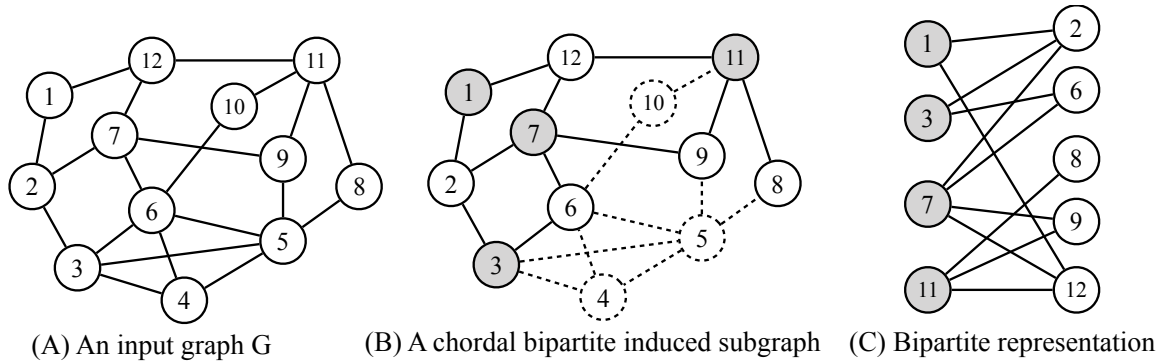


Figure 6.1: (A) shows an input graph G and (B) shows one of the solutions $B = (X, Y, E)$, where $X = \{1, 3, 7, 11\}$ and $Y = \{2, 6, 8, 9, 12\}$. (C) shows the graph B drawn by dividing X and Y.

adjacent. We denote a biclique as $K_{a,b}$ if $|X| = a$ and $|Y| = b$. and say that the size of a biclique $K_{t,t}$ is t. $\mathcal{V}$ is *totally ordered* if for any pair $X, Y \in \mathcal{V}$ of vertex subsets, either $X \subseteq Y$ or $X \supseteq Y$. To show our result, we use the following two theorems.

Theorem 67 (Theorem 1 of [2]). *Let $\mathcal{H}$ be a hypergraph. Then $\mathcal{I}(\mathcal{H})$ is chordal bipartite if and only if $\mathcal{H}$ is β-acyclic.*

Theorem 68 (Theorem 3.9 of [7]). *A β-acyclic hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with at least two vertices has two distinct β-leaves that are not neighbors in $\mathcal{H}' = (\mathcal{V}, \mathcal{E} \setminus \{\mathcal{V}\})$.*

Brault [7] gives a vertex elimination ordering $(v_1, \ldots, v_n)$ for a hypergraph $\mathcal{H}$, called a *β-elimination ordering*. The definition of the ordering is as follows: For any $1 \leq i \leq n$, v_i is a β-leaf in $\mathcal{H}[V_{i\leq}]$, where $V_{i\leq} = \{v_k \in V \mid i \leq k \leq n\}$. Similarly, $V_{\leq i}$ is defined by $\{v_k \in V \mid 1 \leq k \leq i\}$. He also showed that $\mathcal{H}$ is β-acyclic if and only if there is a β-elimination ordering of $\mathcal{H}$ [7]. Similarly, for any graph G, we define a new vertex elimination ordering $(v_1, \ldots, v_n)$ for G, called `CBEO`, as follows: For any $1 \leq i \leq n$,

v_i is a weak-simplicial in $G[V_{i\leq}]$. In the remainder of this section, we show that a graph is chordal bipartite if and only if G has `CBEO`. Lemma 69 shows that a β-leaf of a hypergraph is weak-simplicial in its incidence graph.

Lemma 69. *Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph, v be a vertex in $\mathcal{V}$, and v' be a corresponding vertex of v in X of $\mathcal{I}(\mathcal{H})$. Then v is a β-leaf in $\mathcal{H}$ if and only if v' is a weak-simplicial vertex in $\mathcal{I}(\mathcal{H})$.*

Proof. We assume that v is a β-leaf in $\mathcal{H}$. Let v' be the vertex corresponding to v in $\mathcal{I}(\mathcal{H})$. From the definition of a β-leaf, $\mathcal{N}(v)$ is also totally ordered in $\mathcal{I}(\mathcal{H})$. Thus, v is a weak-simplicial vertex in $\mathcal{I}(\mathcal{H})$. We next assume that v' is weak-simplicial in X of $\mathcal{I}(\mathcal{H})$. From the definition, $\mathcal{N}(v)$ is totally ordered. Thus, $\mathcal{H}(v)$ is totally ordered. Therefore, v is a β-leaf in $\mathcal{H}$ and the statement holds. □

From Lemma 69, a β-leaf v of $\mathcal{H}$ corresponds to a weak-simplicial vertex of the incidence graph $\mathcal{I}(\mathcal{H})$. We next show that a chordal bipartite graph has at least one weak-simplicial vertex from Theorem 67, Theorem 68, and Lemma 69.

Lemma 70. *Let $B = (X, Y, E)$ be a chordal bipartite graph with at least two vertices. If there is no vertex v in B such that $N(v) = X$ or $N(v) = Y$, then B has at least two weak-simplicial vertices which are not adjacent.*

Proof. From Theorem 67, Theorem 68, and Lemma 69, if B is chordal bipartite and has no twins, then G has at two weak-simplicial vertices which are not adjacent. We now assume B has twins. We construct B' as follows: Let $\mathcal{T}$ be a set of twins in B. For each twins $T \in \mathcal{T}$, remove all vertices in T except one twin of T. Note that B' is still chordal bipartite since vertex deletion does not destroy chordality. Since B' has no twins, B' has at least two weak-simplicial vertices u and v which are not adjacent.

Since the set inclusion relation between B and B' is same, u and v also weak-simplicial in B. Hence, the statement holds. □

Theorem 71. *Let B be a bipartite graph. B is chordal bipartite if and only if B has* `CBEO`*.*

Proof. From Lemma 70, the only if part holds. We consider the contrapositive of the if part. Suppose that B is not chordal bipartite. Then B has an induced cycle C with length six or more. Since a vertex in C is not weak-simplicial, we cannot eliminate all vertices from B and the statement holds. □

We next show that a vertex v is weak-simplicial in a bipartite graph B if and only if $B[N^{\leq 2}[v]]$ is bipartite chain. The following lemma is used to improve the time complexity of `ECB-IS` in Section 6.4.

Lemma 72. *Let $B = (X, Y, E)$ be a chordal bipartite graph and v be a vertex in B. Then v is weak-simplicial if and only if an induced subgraph $B[N^{\leq 2}[v]]$ is bipartite chain.*

Proof. We assume that $B[N^{\leq 2}[v]]$ is bipartite chain. From the definition, any pair of vertices in $N(v)$ are comparable. Hence, v is weak-simplicial.

We next prove the other direction. We assume that v is weak-simplicial. Let x and y be vertices in $N^2(v)$. If x and y are incomparable in $B[N^{\leq 2}[v]]$, then there are two vertices $z \in N(x) \setminus N(y)$ and $z' \in N(y) \setminus N(x)$. Note that z and z' are neighbors of v. This contradicts that any pair of vertices in $N(v)$ are comparable. Hence, $x, y \in N^2(v)$ are comparable and $B[N^{\leq 2}[v]]$ is bipartite chain. □

Algorithm 12: Enumeration algorithm for all chordal bipartite induced subgraphs

```
Procedure ECB-IS(G)                    // G = (V, E): an input graph
    RecECB-IS((∅, ∅, V, G));
Procedure RecECB-IS(X, WS(X), AWS(X), G)
    Output X and Compute C(X) from AWS(X);
    for v ∈ C(X) do
        Y ← X ∪ {v};
        if P(Y) = X then RecECB-IS(Y, WS(Y), AWS(Y), G) ;
```

6.3 The proposed algorithm

In this section, we propose an enumeration algorithm `ECB-IS` which is based on reverse search [3]. `ECB-IS` enumerates all solutions by traversing on a tree structure $\mathcal{F}(G) = (\mathcal{S}(G), \mathcal{E}(G))$, called a *family tree*, where $\mathcal{S}(G)$ is a set of solutions in an input graph G and $\mathcal{E}(G) \subseteq \mathcal{S}(G) \times \mathcal{S}(G)$. Note that $\mathcal{F}(G)$ is directed. More precisely, $\mathcal{E}(G)$ is deinfed by the parent-child relationship among solutions based on Theorem 71. Let X be a vertex subset that induces a solution. We denote the set of weak-simplicial vertices in $G[X]$ as $WS(X)$. In what follows, we number the vertex indices from 1 to n and compare the vertices with their indices, where n is the number of vertices in G. The *parent* of X is defined as $\mathcal{P}(X) = X \setminus \{\arg\max WS(X)\}$. X is a *child* of Y if $\mathcal{P}(X) = Y$. Let $ch(X)$ be the set of children of X. We define the *parent vertex* $pv(X)$ as $\arg\max WS(X)$ which induces the parent. For any pair of solutions X and Y, $(X, Y) \in \mathcal{E}(G)$ if $Y = \mathcal{P}(X)$. From Theorem 71, any solution can reach the empty set by recursively removing the parent vertex from the solution. Hence, the following

lemma holds.

Lemma 73. *The family tree forms a tree.*

Next, we show that `ECB-IS` enumerates all solutions. For any vertex subset $X \subseteq V$, we denote $X_{\leq i} = X \cap V_{\leq i}$. An *addible weak-simplicial vertex set* is $AWS(X) = \{v \in V \setminus X \mid v \in WS(X \cup \{v\})\}$, that is, any vertex v in $AWS(X)$ generates a solution $X \cup \{v\}$. We define a *candidate set* $C(X)$ as follows: $C(X) = AWS(X)_{v\leq} \cup (AWS(X) \cap N^{\leq 2}(v))$, where $v = pv(X)$. Note that $C(X)$ is a subset of $AWS(X)$. The following lemma shows that we can enumerate all children if we have $C(X)$.

Lemma 74. *Let X and Y be distinct solutions. If Y is a child of X, then $pv(Y) \in C(X)$.*

Proof. Suppose that Y is a child of X. Let $v = pv(Y)$ and $u = pv(X)$. Note that v belongs to $AWS(X)$. If $u < v$, then $v \in AWS_{u\leq}(X)$ and thus $v \in C(X)$. Otherwise, u is not contained in $WS(Y)$ since v has the maximum index in $WS(Y)$. From the definition of a weak-simplicial vertex, there are two vertices in $N_Y(u)$ which are incomparable in $G[Y]$. Since $\mathcal{N}(u)$ is totally ordered in $G[X]$, v must be in $N_Y^{\leq 2}(u)$. Hence, the statement holds. □

In what follows, we say a vertex $v \in C(X)$ *generates a child* if $X \cup \{v\}$ is a child of X. From Lemma 73 and Lemma 74, `ECB-IS` enumerates all solution by the DFS traversing on $\mathcal{F}(G)$.

Theorem 75. `ECB-IS` *enumerates all solutions without duplication.*

Algorithm 13: Update algorithms for WS and AWS

```
Procedure UpdateWS(X, v, WS(X), G)
    WS ← WS(X);
    for u ∈ N_X(v) do
        if u ∈ WS∧ there is a vertex w ∈ N_X(u) is incomparable to u. then
          WS ← WS \ {u};
        for w ∈ N_X(u) ∩ WS do
            if w and v are incomparable then  WS ← WS \ {w} ;
    return WS;
Procedure UpdateAWS(X, v, AWS(X), G)
    AWS ← AWS(X);
    for u ∈ N(v) do
        if u ∈ AWS then
            if There is a vertex w ∈ N_X(u) which is incomparable to u. then
              AWS ← AWS \ {u};
        else if u ∈ X then
            for w ∈ N(u) ∩ AWS do
                if w and v are incomparable. then  AWS ← AWS \ {w} ;
    return AWS;
```

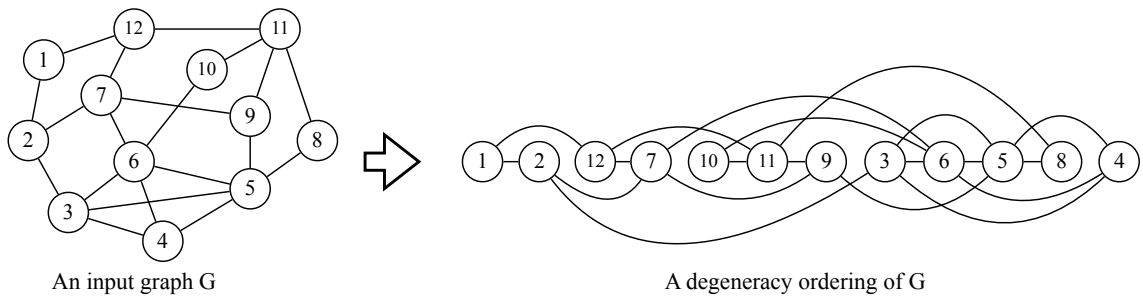


Figure 6.2: It is a degeneracy ordering of G. The degeneracy of G is three. In this ordering, $\left|N^{\leq 2}_{\leq v}(v)\right|$ is at most $k\Delta$ for any vertex v.

6.4 The time complexity analysis

Our proposed algorithm `ECB-IS` has two bottlenecks. (1) Some vertices in $C(X)$ do not generate a child and (2) the maintenance of $WS(X)$ and $AWS(X)$ consumes time. A trivial bound of the number of redundant vertices in $C(X)$ is $O(\Delta^2)$ since only vertices in $(AWS(X) \cap N^{\leq 2}(pv(X))$ may not generate a child, where Δ is the maximum degree of an input graph. To reduce the number of such redundant vertices, we use a *degeneracy ordering*. A graph G is *k-degenerate* if any induced subgraph of G has a vertex with degree k or less [48]. The *degeneracy* of a graph is the smallest such number k. Matula and Beck [53] show that a k-degenerate graph G has a following vertex ordering, called a *degeneracy ordering*: For each vertex v, the number of neighbors smaller than v is at most k. See Figure 6.2. They also show that a k-degenerate ordering of G can be obtained in linear time. Note that there can be several degeneracy orderings for a graph. In what follows, we fix the reverse ordering of a degeneracy ordering and $WS(X)$ and $AWS(X)$ are sorted in this ordering. We first show that the number of redundant vertices is at most $2k\Delta$.

Lemma 76. *Let X be a solution. The number of vertices in $C(X)$ which do not*

generate a child is at most $2k\Delta$.

Proof. Let v be a vertex in $C(X)$ and p be a vertex $pv(X)$. If $p < v$, then v generates a child. We assume that $v < p$. Since v is in $C(X)$, $v \in N^{\leq 2}(p) \cap AWX_{\leq p}(X)$. We estimate the size of $N^{\leq 2}(p) \cap AWX_{\leq p}(X)$. We consider a vertex $u \in N_{\leq v}(v)$. $\sum_{u \in N_{\leq v}(v)} |N(u)|$ is at most $k\Delta$ since $|N_{\leq v}(v)|$ is at most k. We next consider a vertex $u \in N_{v\leq}(v)$. Since u is larger than v, a vertex in $N_{u\leq}(u)$ is larger than v. Hence, we consider vertices $N_{\leq u}(u)$. For each u, $|N_{\leq u}(u)|$ is at most k. Hence, $\sum_{u \in N_{v\leq}(v)} |N(u)_{\leq u}|$ is at most $k\Delta$ and the statement holds. □

We next show how to compute $C(Y)$ from $C(X)$, where X is a solution and Y is a child of X. From the definition of $C(X)$, we can compute $C(Y)$ in $O(|C(Y)| + k\Delta)$ time if we have $AWS(Y)$ and $pv(Y)$. Moreover, if we have $WS(X \cup \{v\})$, then we can determine whether $X \cup \{v\}$ is a child of X or not in constant time since $WS(X \cup \{v\})$ is sorted. Hence, to obtain the children of X, computing $AWS(X)$ and $WS(X)$ dominates the computation time of each iteration. Here, we define two vertex sets as follows:

$$Del_{\mathrm{W}}(X, v) = \{u \in N^{\leq 2}(v) \cap WS(X) \mid u \notin WS(X \cup \{v\})\} \text{ and}$$
$$Del_{\mathrm{A}}(X, v) = \{u \in N^{\leq 2}(v) \cap AWS(X) \mid u \notin WS(X \cup \{u, v\})\}.$$

These vertex sets are the sets of vertices that are removed from $WS(X)$ and $AWS(X)$ after adding v to X, respectively. In the following lemmas, we show that $WS(X)$ and $AWS(X)$ can be updated if we have $Del_{\mathrm{W}}(X, v)$ and $Del_{\mathrm{A}}(X, v)$.

Lemma 77. *Let X be a solution, Y be a child of X, and $v = pv(Y)$. Then $WS(Y) = (WS(X) \setminus Del_W(X, v)) \cup \{v\}$.*

Proof. Let u be a vertex in $WS(Y)$. We prove u is contained in $(WS(X) \backslash Del_{\mathrm{W}}(X, v)) \cup \{v\}$. If $u = v$ holds, then $u \in WS(Y)$ since $v \in pv(Y)$. We assume that $u \neq v$. Since u is weak-simplicial in Y, $u \in WS(X)$. If $u \in Del_{\mathrm{W}}(X, v)$, then u is not weak-simplicial in Y. This contradicts the assumption. Hence, $u \notin Del_{\mathrm{W}}(X, v)$. We prove the other direction. Let u be a vertex in $(WS(X) \setminus Del_{\mathrm{W}}(X, v)) \cup \{v\}$. We assume that $u \neq v$. If $u \notin WS(Y)$, then $u \in Del_{\mathrm{W}}(X, v)$. This is a contradiction and thus the statement holds. □

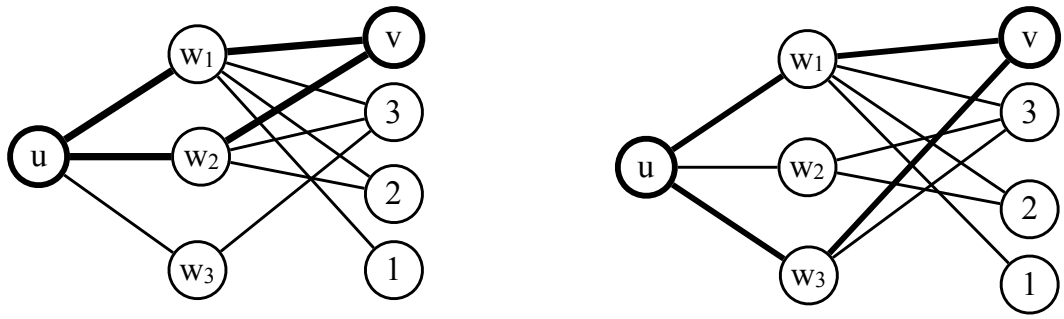
Lemma 78. *Let X be a solution, Y be a child of X, and $v = pv(Y)$. Then $AWS(Y) = AWS(X) \setminus Del_A(X, v)$.*

Proof. Let u be a vertex in $AWS(Y)$. We prove u is contained in $AWS(X) \backslash Del_{\mathrm{A}}(X, v)$. From the definition of $AWS(X)$, $u \in AWS(X)$ holds. If $u \in Del_{\mathrm{A}}(X, v)$, then u is not weak-simplicial in $Y \cup \{u\}$. Since $u \in AWS(Y)$, $u \notin Del_{\mathrm{A}}(X, v)$. We prove the other direction. Let u be a vertex in $AWS(X) \setminus Del_{\mathrm{A}}(X, v)$. From the definition of $AWS(X)$ and $Del_{\mathrm{A}}(X, v)$, u is weak-simplicial in $X \cup \{v, u\}$. Hence, $u \in AWS(Y)$ and the statement holds. □

Note that by just removing redundant vertices, $WS(Y)$ and $AWS(X)$ can be easily sorted if $WS(X)$ and $AWS(X)$ were already sorted. We next consider how to compute $Del_{\mathrm{W}}(X, v)$ and $Del_{\mathrm{A}}(X, v)$. We first show a characterization of a vertex in $Del_{\mathrm{W}}(X, v)$ and $Del_{\mathrm{A}}(X, v)$. In the following lemmas, let X be a solution, v be a vertex in $AWS(X)$, and Y be a solution $X \cup \{v\}$.

Lemma 79. *Let u be a vertex in $N_Y(v) \cap WS(X)$. Then u is contained in $Del_W(X, v)$ if and only if $N_X(u)$ contains w which is incomparable to v.*

Proof. If u has a neighbor w in X which is incomparable to v, then from the definition, u is not weak-simplicial.



(A) When a vertex v is added, u is a weak-simplicial

(B) When a vertex v is added, u is not weak-simplicial

Figure 6.3: Let X be a set of vertices $\{u, w_1, w_2, w_3, 1, 2, 3\}$. In case (A), a vertex u is still weak-simplicial. In case (B), however, u is not weak-simplicial since w_1 and w_3 are imcomparable.

Let u be a vertex in $Del_{\mathrm{W}}(X, v)$. There are vertices w_1 and w_2 in $N_Y(u)$ which are incomparable. If w_1 or w_2 is equal to v, then the statement holds. Hence, we assume that both w_1 and w_2 are not equal to v. Since $G[Y]$ is bipartite, w_1 and w_2 are not adjacent to v. Hence, $N_X(w_1) = N_Y(w_1)$ and $N_X(w_2) = N_Y(w_2)$ hold. This contradicts that X is a solution and the statement holds. □

Lemma 80. *Let u be a vertex in $N_Y^2(v) \cap WS(X)$. Then u is contained in $Del_W(X, v)$ if and only if there exist vertices $w_1, w_2 \in N_X(u)$ which satisfy $N_X(w_1) \subset N_X(w_2)$, $v \in N_Y(w_1)$, and $v \notin N_Y(w_2)$.*

Proof. The if part is easily shown by the assumption of the incomparability of w_1 and w_2. We next prove the other direction. We assume that $u \in Del_{\mathrm{W}}(X, v)$. Hence, there are neighbors w_1 and w_2 of u such that they are incomparable in Y. Without loss of

generality, $N_X(w_1) \subset N_X(w_2)$ holds since u is in $WS(X)$. If $N_X(w_1) = N_X(w_2)$, then w_1 and w_2 are comparable in Y. Since w_1 and w_2 are incomparable in Y, w_1 is adjacent to v and w_2 is not adjacent to v. Thus, the statement holds. □

Lemma 81. *Let u be a vertex in $N(v) \cap AWS(X)$ and $Z = X \cup \{u, v\}$. Then $u \in Del_A(X, v)$ if and only if u has a neighbor $w \in Z$ that is incomparable to v.*

Proof. The if part is trivial from the definition of weak-simplicial. We prove the only if part. We assume that $u \in Del_{\mathrm{A}}(X, v)$ holds. Since $u \in Del_{\mathrm{A}}(X, v)$, u has neighbors w_1 and w_2 which are incomparable in Z. If w_1 or w_2 is equal to v, then u has a neighbor w which is incomparable to v and the statement holds. We next assume that w_1 and w_2 are distinct from v. v is not adjacent to w_1, w_2, or both of them since $G[Y]$ is bipartite. Hence, w_1 and w_2 are comparable in Z since w_1 and w_2 are comparable in $G[X]$. This contradicts that w_1 and w_2 are incomparable in Z and the statement holds. □

Lemma 82. *Let u be a vertex in $N^2(v) \cap AWS(X)$ and $Z = X \cup \{u, v\}$. Then u is contained in $Del_A(X, v)$ if and only if there exists vertices $w_1, w_2 \in N_Z(u)$ which satisfy $N_Z(w_1) \subset N_Z(w_2)$, $v \in N_Z(w_1)$, and $v \notin N_Z(w_2)$.*

Proof. From the assumption, w_1 and w_2 are incomparable in Z. Hence, the if part holds. We prove the other direction. We assume that $u \in Del_{\mathrm{A}}(X, v)$. Hence, u has vertices w_1' and w_2' which are incomparable in Z. Without loss of generality, $N_{X \cup \{u\}}(w_1') \subset N_{X \cup \{u\}}(w_2')$ holds. Since w_1' and w_2' are incomparable in $G[Z]$, w_1' is adjacent to v. Thus, the statement holds. □

Next, we consider the time complexity of computing $Del_{\mathrm{W}}(X, v)$ and $Del_{\mathrm{A}}(X, v)$. For analysing these computing time more precisely, we give two upper bounds with respect to the number of edges and the size of $N^2(v)$ in bipartite chain graphs. Note that t is the maximum size of a biclique $K_{t,t}$ that appears in B as an induced subgraph.

Lemma 83. *Let B be a bipartite chain graph and v be a vertex in B. Then the size of $N^2(v)$ is at most Δ.*

Proof. Let u be a vertex in $N(v)$ which satisfies $N(w) \subseteq N(u)$ for any $w \in N(v)$. Since $N^2(v) = \bigcup_{w \in N(v)} N(w)$, $N^2(v) = N(u)$. Hence, the statement holds. □

Lemma 84. *Let $B = (X, Y, E)$ be a bipartite chain graph. Then the number of edges in B is $O(t\Delta)$.*

Proof. Let w be a vertex in X which satisfies for $N(u) \supseteq N(w)$ for any $u \in X$. If $d(w) \leq t$, then the statement holds from Lemma 83. We assume that $d(w) > t$. We consider the number of edges in B. Let $(u_1, \ldots, u_{d(w)})$ be a sequence of vertices in $N(w)$ such that $N(u_i) \subseteq N(u_{i+1})$ for $1 \leq i < d(w)$. For each $d(w) - t + 1 \leq i \leq d(w)$, the sum of $|N(u_i)|$ is at most $O(t\Delta)$. We next consider the case for $1 \leq i \leq d(w) - t$. Since $N(u_i)$ is a subset of $N(u_j)$ for any $i < j$, $|N(u_i)|$ is at most t. If $|N(u_i)|$ is greater than t, then B has a biclique $K_{t+1,t+1}$. Hence, the number of edges in B is $O(t\Delta)$ and the statement holds. □

Lemma 85. *Let v be a vertex in $C(X)$. Then we can compute $Del_W(X, v)$ in $O(t\Delta)$ time.*

Proof. Let Y be a solution $X \cup \{v\}$. We first compute vertices in $WS(X) \cap N_X^2(v)$ that remain in $WS(Y)$. From Lemma 77 and Lemma 80, $w \notin WS(Y)$ if and only if there exists vertices $w_1, w_2 \in N(u)$ which satisfy $N_X(w_1) \subset N_X(w_2)$, $v \in N_Y(w_1)$, and $v \notin N_Y(w_2)$. By scanning vertices with distance two from v, this can be done in linear time in the size of $\sum_{u \in N_X(v)} |N_X(u)|$. Since $G[N^{\leq 2}(v)]$ is bipartite chain from Lemma 72, $\sum_{u \in N_X(v)} |N_X(u)|$ is at most $O(t\Delta)$ from Lemma 84. Moreover, it can be determined whether $w \in N^2(v)$ and v are comparable or not in this scan operation.

We next compute vertices in $WS(X) \cap N_X(v)$ that remain in $WS(Y)$. From Lemma 79, u is contained in $Del_{\mathrm{W}}(X, v)$ if and only if u has a neighbor w which is incomparable to v. Since $G[Y]$ is bipartite, $N_X(u)$ is contained in $N_Y^2(v)$. In the previous scan operation, we already know whether w and v are comparable or not. Hence, we can compute whether $w \in WS(Y)$ or not in $O(|N(u)|)$ time. Since $O\left(\sum_{u \in N_Y(v)} |N(u)|\right) = O(t\Delta)$ holds from Lemma 84, we can find $N_X(v) \cap Del_{\mathrm{W}}(X, v)$ in $O(t\Delta)$ total time. Hence, the statement holds. □

Lemma 86. *Let v be a vertex in $C(X)$. Then we can compute $Del_A(X, v)$ in $O(\Delta^2)$ time.*

Proof. Since v is contained in $Del_{\mathrm{A}}(X, v)$, $G[X \cup \{v\}]$ is a chordal bipartite induced subgraph. Here, let Y be a set of vertices $X \cup \{v\}$. In the same fashion as Lemma 85, we can decide $u \in AWS(X \cup \{v\} \cap N_Y^2(v))$ in $O(\Delta^2)$ total time. By applying the above procedure for all vertices distance two from v, we can obtain all vertices in $Del_{\mathrm{A}}(X, v)$ in $O(\Delta^2)$ time since the number of edges can be bounded in $O(\Delta^2)$. □

From Lemma 85 and Lemma 86, we can compute $Del_{\mathrm{W}}(X, v)$ and $Del_{\mathrm{A}}(X, v)$ in $O(t\Delta)$ and $O(\Delta^2)$ time for each $v \in C(X)$, respectively.

Hence, we can enumerate all children in $O(|C(X)|\, t\Delta + |ch(X)|\, \Delta^2)$ time from Lemma 74, Lemma 77 and Lemma 78. In the following theorem, we show the amortized time complexity and the space usage of `ECB-IS`.

Theorem 87. `ECB-IS` *enumerates all solutions in amortized $O(kt\Delta^2)$ time by using $O(n + m)$ space, where k is the degeneracy of an input graph G, t is the maximum size of $K_{t,t}$ that appears in G as an induced subgraph, n is the number of vertices of G, and m is the number of edges of G.*

Proof. `ECB-IS` uses $AWS(X)$ and $WS(X)$ as data structures. Each data structure demands linear space and the total space usage of `ECB-IS` is $O(n+m)$ space. Hence, the total space usage of `ECB-IS` is linear in the size of the input. We next consider the amortized time complexity of `ECB-IS`. From Lemma 75, `ECB-IS` enumerates all solutions. From Lemma 85 and Lemma 86, `ECB-IS` computes all children and updates all data structures in $O(|C(X)|\, t\Delta + |ch(X)|\, \Delta^2)$ time. From Lemma 76, $|C(X)|$ is at most $|ch(X)| + k\Delta$. Hence, we need $O((|ch(X)| + k\Delta)t\Delta + |ch(X)|\, \Delta^2)$ time to generate all children. Note that this computation time is bounded by $O((|ch(X)| + kt)\Delta^2)$. We consider the total time for enumerating all solutions. Since each iteration X needs $O((|ch(X)| + kt)\Delta^2)$ time, the total time is $O\left(\sum_{X\in\mathcal{S}}(|ch(X)| + kt)\Delta^2\right)$ time, where $\mathcal{S}$ is the set of solutions. Since $O\left(\sum_{X\in\mathcal{S}} |ch(X)|\, \Delta^2\right)$ is bounded by $O(|\mathcal{S}|\, \Delta^2)$, the total time is $O(|\mathcal{S}|\, kt\Delta^2)$ time. Therefore, `ECB-IS` enumerates all solutions in amortized $O(kt\Delta^2)$ time. $\square$

Chapter 7

Conclusion and Future Work

In this thesis, we studied efficient enumeration for sparse graphs. In particular, we address the following four enumeration problems: the induced matching enumeration problem, the enumeration of subgraph with bounded girth, the dominating set enumeration problem, and the chordal bipartite induced subgraph enumeration problem. We summarize our results of this thesis.

For the induced matching enumeration problem, we develop a constant amortized time algorithm for C_4-free graphs. The time complexity of a simple binary partition algorithm is $O(\Delta^2)$. The key idea of this algorithm is the maintenance of a candidate set and estimation of the number of descendant. Our algorithm needs $O(\Delta^2)$ time per nodes in an enumeration tree. However, we show that each node has only constant cost.

For the enumeration of subgraph and induced subgraph with bounded girth problem, we consider the problem in directed graphs and undirected graphs. In these problems, we achieve $O(n)$ delay enumeration. More precisely, our algorithm is more efficient than $O(n)$ delay. See for the details in Table 4.1. The bottleneck of this prob-

lem is the computation of girth. For general graphs, the best efficient algorithm needs $O(nm)$ time [32]. However, we overcome this difficulty by using distance matrices.

For the dominating set enumeration problem, we propose two efficient enumeration algorithms. One algorithm enumerates all dominating sets in amortized $O(k)$ time, where k is the degeneracy of a graph. The other algorithm enumerates all dominating sets in constant amortized time for graphs with girth at least nine. The time complexity of a simple enumeration algorithm based on reverse search is amortized $O(\Delta^2)$ time. For bounded degenerate graphs,

For graphs with girth at least nine, each node in an enumeration tree needs $O(\Delta^3)$ time. This time complexity is worse than the time complexity of a simple algorithm. However, we show that each node has many descendants. Thus, we can amortize this computational cost to such descendants. Hence, we can improve the time complexity of this algorithm.

For chordal bipartite induced subgraph enumeration, we propose an amortized $O(kt\Delta^2)$ time enumeration algorithm. The key point of this algorithm is a vertex elimination ordering, called `CBEO`. This ordering is a characterization of chordal bipartite graphs. Our algorithm is based on reverse search and `CBEO` is important in the parent child relation. In our algorithm, we manage a set of weak-simplicial vertices and a set of addible weak-simplicial vertices in $O(t\Delta)$ time and $O(\Delta^2)$ time, respectively. Since the number of redundant children is at most $O(k\Delta)$, our algorithm runs in amortized $O(kt\Delta^2)$ time.

We show future directions of this research area. Firstly, we will develop a graph search algorithm based on enumeration algorithm. Binary partition based algorithms are typically simple. Hence, it is easy to combine other pruning techniques. Indeed, Ajami and Cohen [1] propose top-k minimal set cover enumeration algorithm with

an approximate order. This algorithm is based on simple binary partition. As other application of these algorithms, we consider enumeration of all solutions with the size more than k. For example, we consider enumeration of induced matchings with the size k. By using our algorithm in Chapter 3, we can enumerate all solutions. Since our algorithm is simple backtracking, we add a pruning operation. By computing an upper bound of the size of maximum induced matching, we prune redundant iterations. This algorithm is not output sensitive. However, practically faster than the original algorithm.

Secondly, we will develop enumeration algorithm for maximal or minimal solutions. Our proposed algorithms enumerate all solutions efficiently. However, the number of solution is huge. Hence, we should consider how to reduce the number of solution since the total computation time is long. For that purpose, enumeration of maximal or minimal solution is appropriate. Indeed, the maximal induced matching enumeration problem is similar to maximal independent set enumeration problem. Hence, we would enumerate all maximal induced matchings by applying similar technique to maximal independent set enumeration.

Finally, we will implement theoretical efficient enumeration algorithms. For output sensitive enumeration algorithms, the number of solution is a big problem. If we can develop an efficient counting algorithm, then we can estimate the running time of output sensitive algorithm. Otherwise, we do not know how much time is needed for the computation. Therefore, it is important to know how large and dense data can be enumerated in a realistic time. Since the number of solution is typically huge, it is necessary to reduce the number of solutions, i.e., output only maximal or minimal solutions. For that reason, development and implementation on maximal or minimal subgraph enumeration algorithms and experiments on real data is a future work.

Bibliography

[1] Zahi Ajami and Sara Cohen. Enumerating minimal weight set covers. In *2019 IEEE 35th International Conference on Data Engineering, ICDE 2019*, pages 518–529. IEEE, 2019.

[2] Giorgio Ausiello, Alessandro D'Atri, and Marina Moscarini. Chordality properties on graphs and minimal conceptual connections in semantic data models. *Journal of Computer and System Sciences*, 33(2):179–202, 1986.

[3] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1–3):21–46, 1996.

[4] Etienne Birmelé, Rui Ferreira, Roberto Grossi, Andrea Marino, Nadia Pisanti, Romeo Rizzi, and Gustavo Sacomoto. Optimal Listing of Cycles and st-Paths in Undirected Graphs. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, Louisiana, USA, January 68, 2013.*, pages 118–128. SIAM, 2013.

[5] Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. Generating maximal independent sets for hypergraphs with bounded edge-intersections. In Martin Farach-Colton, editor, *LATIN 2004: Theoretical Informatics - 6th Latin*

American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings, pages 488–498, Berlin, Heidelberg, 2004. Springer-Verlag.

[6] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.

[7] Johann Brault-Baron. Hypergraph acyclicity revisited. *ACM Computing Surveys*, 49(3):54, 2016.

[8] L. Sunil Chandran, Louis Ibarra, Frank Ruskey, and Joe Sawada. Generating and characterizing the perfect elimination orderings of a chordal graph. *Theoretical Computer Science*, 307(2):303–317, 2003.

[9] Lijun Chang, Jeffrey Xu Yu, and Lu Qin. Fast maximal cliques enumeration in sparse graphs. *Algorithmica*, 66(1):173–186, 2013.

[10] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal of Computing*, 14(1):210–223, 1985.

[11] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *Journal of Computer and System Sciences*, 74(7):1147 – 1159, 2008.

[12] Alessio Conte, Donatella Firmani, Caterina Mordente, Maurizio Patrignani, and Riccardo Torlone. Fast enumeration of large k-plexes. In *Proceedings of the 23rd AMC SIGKDD International Conference on Knowledge Discovery and Data Mining Halifax, NS, Canada - August 13 - 17, 2017*, pages 115–124. ACM, 2017.

[13] Alessio Conte, Roberto Grossi, Andrea Marino, and Romeo Rizzi. Efficient enumeration of graph orientations with sources. *Discrete Applied Mathematics*, 246:22–37, 2018.

[14] Alessio Conte, Roberto Grossi, Andrea Marino, Takeaki Uno, and Luca Versari. Listing maximal independent sets with minimal space and bounded delay. In Gabriele Fice, Marinella Sciortino, and Rossano Venturini, editors, *String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017, Palermo, Italy, September 2629, 2017, Proceedings*, volume 10508 of *Lecture Notes in Computer Science*, pages 144–160, Cham, Switzerland, 2017. Springer International Publishing.

[15] Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. Sublinear-space bounded-delay enumeration for massive network analytics: Maximal cliques. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *Leibniz International Proceedings in Informatics (LIPcs)*, pages 148:1–148:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.

[16] Alessio Conte, Kazuhiro Kurita, Kunihiro Wasa, and Takeaki Uno. Listing acyclic subgraphs and subgraphs of bounded girth in directed graphs. In Xiaofeng Gao, Hongwei Du, and Meng Han, editors, *Combinatorial Optimization and Applications - 11th International Conference, COCOA 2017, Shanghai, China, December 16-18, 2017, Proceedings, Part II*, volume 10628 of *Lecture Notes in Computer Science*, pages 169–181, Cham, Switzerland, 2017. Springer International Publishing.

[17] Alessio Conte and Takeaki Uno. New polynomial delay bounds for maximal subgraph enumeration by proximity search. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, Phoenix, AZ, USA - June 23 - 26, 2019*, pages 1179–1190. ACM, 2019.

[18] Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 157(12):2675–2700, 2009.

[19] Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM Journal on Computing*, 32(2):514–537, 2003.

[20] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. Efficient densest subgraph computation in evolving graphs. In Aldo Gangemi, Stefano Leonardi, and Alessandro Panconesi, editors, *Proceedings of the 24th International Conference on World Wide Web, Florence, Italy - May 18 - 22, 2015*, WWW '15, pages 300–310, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee.

[21] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *Journal of Experimental Algorithmics*, 18:1–21, 2013.

[22] Martin Farber. Characterizations of strongly chordal graphs. *Discrete Mathematics*, 43(2-3):173–189, 1983.

[23] Rui Ferreira. *Efficiently Listing Combinatorial Patterns in Graphs*. PhD thesis, Universitá degli Studi di Pisa, 2013.

[24] Rui Ferreira, Roberto Grossi, and Romeo Rizzi. Output-sensitive listing of bounded-size trees in undirected graphs. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 of *Lecture Notes in Computer Science*, pages 275–286, Berlin, Heidelberg, 2011. Springer-Verlag.

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

[26] Alain Gély, Lhouari Nourine, and Bachir Sadi. Enumeration aspects of maximal cliques and bicliques. *Discrete Applied Mathematics*, 157(7):1447–1459, 2009.

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

[28] 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.

[29] 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.

[30] Roberto Grossi, Andrea Marino, and Luca Versari. Efficient algorithms for listing k disjoint st-paths in graphs. In Michael A. Bender, Martin Farach-Colton, and Miguel A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics 13th*

Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings, volume 10807 of *Lecture Notes in Computer Science*, pages 544–557, Cham, Switzerland, 2018. Springer International Publishing.

[31] Jing Huang. Representation characterizations of chordal bipartite graphs. *Journal of Combinatorial Theory, Series B*, 96(5):673–683, 2006.

[32] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.

[33] David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119 – 123, 1988.

[34] Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. Enumeration of minimal dominating sets and variants. In Olaf Owe, Martin Steffen, and Jan Arne Telle, editors, *Fundamentals of Computation Theory - 18th International Symposium, FCT 2011, Oslo, Norway, August 22-28, 2011, Proceedings*, volume 6914 of *Lecture Notes in Computer Science*, pages 298–309, Berlin, Heidelberg, 2011. Springer-Verlag.

[35] Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM Journal on Discrete Mathematics*, 28(4):1916–1929, 2014.

[36] 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 Ernst W. Mayr, editor, *Graph-Theoretic Concepts*

in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 2729, 2018, Proceedings, volume 11159 of *Lecture Notes in Computer Science*, pages 138–153, Berlin, Heidelberg, 2015. Springer-Verlag.

[37] Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, Lhouari Nourine, and Takeaki Uno. Polynomial delay algorithm for listing minimal edge dominating sets in graphs. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, volume 9214 of *Lecture Notes in Computer Science*, pages 446–457, Cham, Switzerland, 2015. Springer International Publishing.

[38] Leonid Khachiyan, Endre Boros, Konrad Borys, Khaled Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. Generating cut conjunctions in graphs and related problems. *Algorithmica*, 51(3):239–263, 2008.

[39] Leonid Khachiyan, Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. Enumerating disjunctions and conjunctions of paths and cuts in reliability theory. *Discrete Applied Mathematics*, 155(2):137–149, 2007.

[40] Masashi Kiyomi, Shuji Kijima, and Takeaki Uno. Listing chordal graphs and interval graphs. In Fedor V. Fomin, editor, *Graph-Theoretic Concepts in Computer Science - 32nd International Workshop, WG 2006, Bergen, Norway, June 22-23, 2006, Revised Papers*, volume 4271 of *Lecture Notes in Computer Science*, pages 68–77, Berlin, Heidelberg, 2006. Springer-Verlag.

[41] Masashi Kiyomi and Takeaki Uno. Generating chordal graphs included in given graphs. *IEICE TRANSACTIONS on Information and Systems*, E89-D(2):763–770, 2006.

[42] Christian Komusiewicz and Frank Sommer. Enumerating connected induced subgraphs: Improved delay and experimental comparison. In Barbara Catania, Rastislav Královic, Jerzy R. Nawrocki, and Giovanni Pighizzini, editors, *SOFSEM 2019: Theory and Practice of Computer Science - 45th International Conference on Current Trends in Theory and Practice of Computer Science, Nov Smokovec, Slovakia, January 27-30, 2019, Proceedings*, volume 11376 of *Lecture Notes in Computer Science*, pages 272–284, Cham, Switzerland, 2019. Springer Nature Switzerland.

[43] Kazuhiro Kurita, Kunihiro Wasa, Hiroki Arimura, and Takeaki Uno. Efficient enumeration of dominating sets for sparse graphs. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018 - Jiaoxi, Yilan, Taiwan*, volume 123 of *Leibniz International Proceedings in Informatics (LIPcs)*, pages 8:1–8:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.

[44] Kazuhiro Kurita, Kunihiro Wasa, Alessio Conte, Takeaki Uno, and Hiroki Arimura. Efficient enumeration of subgraphs and induced subgraphs with bounded girth. In Costas S. Iliopoulos, Hon Wai Leong, and Wing-Kin Sung, editors, *Combinatorial Algorithms - 29th International Workshop, IWOCA 2018, Singapore, July 1619, 2018, Proceedings*, volume 10979 of *Lecture Notes in Computer Science*, pages 201–213, Cham, Switzerland, 2018. Springer International Publishing.

[45] Kazuhiro Kurita, Kunihiro Wasa, Takeaki Uno, and Hiroki Arimura. Efficient enumeration of induced matchings in a graph without cycles with length four. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, E101-A(9):1383–1391, 2018.

[46] Kazuhiro Kurita, Kunihiro Wasa, Takeaki Uno, and Hiroki Arimura. An efficient algorithm for enumerating chordal bipartite induced subgraphs in sparse graphs. In Charles J. Colbourn, Roberto Grossi, and Nadia Pisanti, editors, *Combinatorial Algorithms - 30th International Workshop, IWOCA 2019, Pisa, Italy, July 2325, 2019, Proceedings*, volume 11638 of *Lecture Notes in Computer Science*, pages 339–351, Cham, Switzerland, 2019. Springer International Publishing.

[47] E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.

[48] Don R. Lick and Arthur T. White. k-degenerate graphs. *Canadian Journal of Mathematics*, 22(5):1082–1096, 1970.

[49] Anna Lubiw. Doubly lexical orderings of matrices. *SIAM Journal on Computing*, 16(5):854–879, 1987.

[50] Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. In Torben Hagerup and Jyrki Katajainen, editors, *Algorithm Theory - SWAT 2004 - 9th Scandinavian Workshop on Algorithm Theory, Humlebaek, Denmark, July 8-10, 2004. Proceedings*, volume 3111 of *Lecture Notes in Computer Science*, pages 260–272, Berlin, Heidelberg, 2004. Springer-Verlag.

[51] George Manoussakis. A new decomposition technique for maximal clique enumeration for sparse graphs. *Theoretical Computer Science*, 770:25–33, 2019.

[52] Yasuko Matsui, Ryuhei Uehara, and Takeaki Uno. Enumeration of the perfect sequences of a chordal graph. *Theoretical Computer Science*, 411(40-42):3635–3641, 2010.

[53] David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983.

[54] Hannes Moser and Somnath Sikdar. The parameterized complexity of the induced matching problem. *Discrete Applied Mathematics*, 157(4):715–727, 2009.

[55] Shin-ichi Nakano. Enumerating floorplans with n rooms. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, E85-A(7):1746–1750, 2002.

[56] James B. Orlin and Antonio Sedeno-Noda. An $O(nm)$ time algorithm for finding the min length directed cycle in a graph. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, Barcelona, Spain - January 16 - 19, 2017*, pages 1866–1879. SIAM, 2017.

[57] Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Transactions on Algorithms*, 9(1):11, 2012.

[58] J. Scott Provan and Douglas R. Shier. A paradigm for listing (s, t)-cuts in graphs. *Algorithmica*, 15(4):351–372, 1996.

[59] Venkatesh Raman and Saket Saurabh. Short cycles make W-hard problems hard: FPT algorithms for W-hard problems in graphs with no short cycles. *Algorithmica*, 52(2):203–225, 2008.

[60] R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.

[61] Benno Schwikowski and Ewald Speckenmeyer. On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics*, 117(1-3):253–265, 2002.

[62] Akiyoshi Shioura, Akihisa Tamura, and Takeaki Uno. An Optimal Algorithm for Scanning All Spanning Trees of Undirected Graphs. *SIAM Journal on Computing*, 26(3):678–692, 1997.

[63] Larry J. Stockmeyer and Vijay V. Vazirani. NP-completeness of some generalizations of the maximum matching problem. *Information Processing Letters*, 15(1):14–19, 1982.

[64] Ken Takata. Space-optimal, backtracking algorithms to list the minimal vertex separators of a graph. *Discrete Applied Mathematics*, 158(15):1660–1667, 2010.

[65] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977.

[66] Shuji Tsukiyama, Isao Shirakawa, Hiroshi Ozaki, and Hiromu Ariyoshi. An algorithm to enumerate all cutsets of a graph in linear time per cutset. *Journal of the ACM*, 27(4):619–632, 1980.

[67] Ryuhei Uehara. Linear time algorithms on chordal bipartite and strongly chordal graphs. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming - 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002. Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 993–1004, Berlin, Heidelberg, 2002. Springer-Verlag.

[68] Takeaki Uno. An algorithm for enumerating all directed spanning trees in a directed graph. In Tetsuo Asano, Yoshihide Igarashi, Hiroshi Nagamochi, Satoru Miyano, and Subhash Suri, editors, *Algorithms and Computation - 7th International Symposium, ISAAC '96, Osaka, Japan, December 16 - 18, 1996, Proceedings*, volume 1178 of *Lecture Notes in Computer Science*, pages 166–173, Berlin, Heidelberg, 1996. Springer-Verlag.

[69] Takeaki Uno. A fast algorithm for enumerating bipartite perfect matchings. In Peter Eades and Tadao Takaoka, editors, *Algorithms and Computation - 12th International Symposium, ISAAC 2001, Christchurch, New Zealand, December 19-21, 2001. Proceedings*, volume 2223 of *Lecture Notes in Computer Science*, pages 367–379, Berlin, Heidelberg, 2001. Springer-Verlag.

[70] Takeaki Uno. A fast algorithm for enumerating non-bipartite maximal matchings. *NII Journal*, 3:89–97, 2001.

[71] Takeaki Uno. Two general methods to reduce delay and change of enumeration algorithms. Technical report, National Institute of Informatics, 2003.

[72] Takeaki Uno. An efficient algorithm for enumerating pseudo cliques. In Takeshi Tokuyama, editor, *Algorithms and Computation - 18th International Symposium,*

ISAAC 2007, Sendai, Japan, December 17-19, 2007, Proceedings, volume 4835 of *Lecture Notes in Computer Science*, pages 402–414, Berlin, Heidelberg, 2007. Springer-Verlag.

[73] Takeaki Uno. Constant time enumeration by amortization. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Algorithms and Data Structures - 15th International Symposium, WADS 2015, St. John' s, NL, Canada, July 31 August 2, 2017, Proceedings*, volume 9214 of *Lecture Notes in Computer Science*, pages 593–605, Cham, Switzerland, 2015. Springer International Publishing.

[74] Kunihiro Wasa. Enumeration of enumeration algorithms, 2019. `https://kunihirowasa.github.io/enum/`, (visited on 2019-12-14).

[75] Kunihiro Wasa, Hiroki Arimura, and Takeaki Uno. Efficient Enumeration of Induced Subtrees in a K-Degenerate Graph. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation - 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*, volume 8889 of *Lecture Notes in Computer Science*, pages 94–102, Cham, Switzerland, 2014. Springer International Publishing.

[76] Kunihiro Wasa, Yusaku Kaneta, Takeaki Uno, and Hiroki Arimura. Constant time enumeration of bounded-size subtrees in trees and its application. In Joachim Gudmundsson, Julián Mestre, and Taso Viglas, editors, *Computing and Combinatorics - 18th Annual International Conference, COCOON 2012*, volume 7434 of *Lecture Notes in Computer Science*, pages 347–359, Berlin Heidelberg, 2012. Springer-Verlag.

[77] Kunihiro Wasa and Takeaki Uno. An efficient algorithm for enumerating induced subgraphs with bounded degeneracy. In Donghyun Kim, R. N. Uma, and Alexander Zelikovsky, editors, *Combinatorial Optimization and Applications - 12th International Conference, COCOA 2018, Atlanta, GA, USA, December 15-17, 2018, Proceedings*, volume 11346 of *Lecture Notes in Computer Science*, pages 35–45, Cham, Switzerland, 2018. Springer International Publishing.

[78] Kunihiro Wasa and Takeaki Uno. Efficient enumeration of bipartite subgraphs in graphs. In Lusheng Wang and Daming Zhu, editors, *Computing and Combinatorics - 24th International Conference, COCOON 2018, Qing Dao, China, July 2-4, 2018, Proceedings*, volume 10976 of *Lecture Notes in Computer Science*, pages 454–466, Cham, Switzerland, 2018. Springer International Publishing.

[79] Kazuaki Yamazaki, Toshiki Saitoh, Masashi Kiyomi, and Ryuhei Uehara. Enumeration of nonisomorphic interval graphs and nonisomorphic permutation graphs. *Theoretical Computer Science*, 2019. `https://doi.org/10.1016/j.tcs.2019.04.017`.