4-28-2022

Efficient Sampling and Counting of Graph Structures related to Chordal Graphs

Wenbo Sun
ws3109@rit.edu

Efficient Sampling and Counting of Graph Structures related to Chordal Graphs

by

Wenbo Sun

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in Computing and Information Sciences

B. Thomas Golisano College of Computing and
Information Sciences

Rochester Institute of Technology
Rochester, New York
April 28, 2022

Efficient Sampling and Counting of Graph Structures related to Chordal Graphs

by

Wenbo Sun

Committee Approval:

We, the undersigned committee members, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements of the degree of Doctor of Philosophy in Computing and Information Sciences.

Ivona Bezáková — Date
Dissertation Advisor

Edith Hemaspaandra — Date
Dissertation Committee Member

Carlos R. Rivero — Date
Dissertation Committee Member

Daniel Štefankovič — Date
Dissertation Committee Member

Anurag Agarwal — Date
Dissertation Defense Chairperson

Certified by:

Pengcheng Shi — Date
Ph.D. Program Director, Computing and Information Sciences

Efficient Sampling and Counting of Graph Structures related to Chordal Graphs

by

Wenbo Sun

Submitted to the
B. Thomas Golisano College of Computing and Information Sciences Ph.D. Program in Computing and Information Sciences
in partial fulfillment of the requirements for the
Doctor of Philosophy Degree
at the Rochester Institute of Technology

Abstract

Counting problems aim to count the number of solutions for a given input, for example, counting the number of variable assignments that satisfy a Boolean formula. Sampling problems aim to produce a random object from a desired distribution, for example, producing a variable assignment drawn uniformly at random from all assignments that satisfy a Boolean formula. The problems of counting and sampling of graph structures on different types of graphs have been studied for decades for their great importance in areas like complexity theory and statistical physics. For many graph structures such as independent sets and acyclic orientations, it is widely believed that no exact or approximate (with an arbitrarily small error) polynomial-time algorithms on general graphs exist. Therefore, the research community studies various types of graphs, aiming either to design a polynomial-time counting or sampling algorithm for such inputs, or to prove a corresponding inapproximability result. Chordal graphs have been studied widely in both AI and theoretical computer science, but their study from the counting perspective has been relatively limited. Previous works showed that some graph structures can be counted in polynomial time on chordal graphs, when their counting on general graphs is provably computationally hard. The main objective of this thesis is to design and analyze counting and sampling algorithms for several well-known graph structures, including independent sets and different types of graph orientations, on chordal graphs. Our contributions can be described from two perspectives: evaluating the performances of some well-known sampling techniques, such as Markov chain Monte Carlo, on chordal graphs; and showing that the chordality does make those counting problems polynomial-time solvable.

Acknowledgments

The work in this thesis was partially supported by grant no. DUE-1819546 of the National Science Foundation.

Contents

List of Figures

Chapter 1

Introduction

In computational complexity theory, a decision problem is a problem that can be viewed as a yes-or-no property of the input. For instance, deciding whether an integer x is divisible by an integer y is a decision problem. Class P is a complexity class of decision problems that can be solved by a deterministic Turing machine in polynomial time, and NP is a complexity class of decision problems where for each yes-input there is a "proof" attesting that the answer is "yes," which can be verified by a deterministic Turing machine in polynomial time. It follows that P is a subset of NP. The hardest problems in NP are classified as NP-complete problems which are widely believed to be unlikely to be solvable in polynomial time—this relates to the famous "P=NP?" problem in computer science. A classical NP-complete problem is the independent set problem [39]: for a given graph G and a positive integer k, decide whether there is an independent set of size k in G (an independent set of a graph is a set of vertices with no edges between them). For the formal definitions of classes P, NP, and NP-complete, see, for example [65].

The complexity class #P (pronounced "sharp P" or "number P") [74] is a class of the counting problems associated with the decision problems in NP. For instance, the problem of counting independent sets of a given size in a graph is in #P. We will refer to the counting version of a problem by adding a "#" in front of it, such as #independent-sets. Analogously to NP-completeness, the hardest problems in #P are #P-complete. It is generally believed that the counting version of every NP-complete problem is #P-complete (at least, to the best of our knowledge, there are no known counterexamples to this statement). Interestingly, the counting version of a problem in P could still be #P-complete [74]. #P problems that can be solved exactly in polynomial time are relatively rare, two classical and perhaps surprising ones are counting spanning trees in general graphs [42]

and counting perfect matchings in planar graphs [40]. Even if solving a counting problem exactly in polynomial time is infeasible, it might still be possible to solve it approximately in polynomial time, i.e., get an approximate count which is arbitrarily close to the correct count in time that is polynomial in the size of the input and the closeness of the approximation. A polynomial-time approximate counter is called a *fully polynomial time approximation scheme (FPTAS)* or a *fully polynomial time randomized approximation scheme (FPRAS)*, depending on whether the algorithm is randomized or not. For instance, Jerrum and Sinclair [34, 36] designed an FPRAS to count all matchings in general graphs (a matching is a set of edges with no shared end-points), and, in a joint work with Vigoda, they also obtained an FPRAS to count perfect matchings in bipartite graphs [37]. For a counting problem, the research community is interested in either finding an FPTAS (FPRAS) or proving its inapproximability.

Different methods have been used for approximate counting, and, for so-called *self-reducible* problems [38], approximate counting can be achieved by (almost) uniform sampling [38]. In fact, (almost) uniform sampling and approximate counting are polynomially-equivalent (that is, one can be converted to the other in polynomial time) when the problem is self-reducible. Therefore, we can focus on designing (almost) uniform sampling algorithms for self-reducible problems. Markov chain Monte Carlo techniques have been successfully used to sample graph structures like matchings or independent sets according to the so-called Gibbs distribution (which, roughly, gives sampling preference according to the object's energy—for graph structures, this is typically the size of the object), see, for example, [4,16,19,36,47]. There are also other sampling techniques like correlation decay and partial rejection sampling [27, 80].

One important application of approximate counting is estimating the Tutte polynomial of a graph. The Tutte polynomial $T_G(x, y)$ of an undirected graph G is a graph polynomial, whose definition depends on G and will be introduced formally later with two variables x and y. It has been widely studied in graph theory for its close relationship to many important graph quantities, such as #colorings or the chromatic polynomial ($T_G(x, 0)$ [73]), #acyclic-orientations ($T_G(2, 0)$ [82]), and #strong-orientations ($T_G(0, 2)$ [44]). It is also a research topic in statistical physics because it generalizes famous models like the Ising model, the Potts model, and the hard-core model (also known as weighted counting or sampling of independent sets in a graph [82]). Unfortunately, calculating the Tutte polynomial is generally #P-complete. So, many researches focus on finding parameters x, y that make the calculation feasible. For instance, when x and y lie on the hyperbola $(x - 1)(y - 1) = 2$ in the plane, the Tutte polynomial becomes the partition function of the Ising model, for which an FPRAS exists under some mild conditions [35]. Besides, calculation of the Tutte polynomial on different types of graphs, such as bipartite graphs [78] and planar graphs [77],

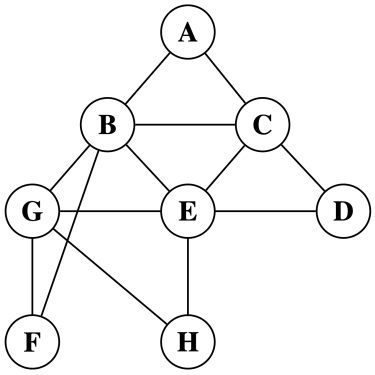

Figure 1.1: A chordal graph G. Cycle ABCD has a chord BC, cycle BEHG has a chord EG, and cycle BCDE has a chord CE.

has also attracted lots of attention.

A graph is *chordal* if all its cycles of length at least four have an edge (a chord) that connects two non-adjacent vertices on the cycle (see Figure 1.1).

Chordal graphs have attracted great attention in computer science theory as a natural graph class with real-world applications (for example, some inference techniques in probabilistic graphical models rely on sampling and counting of certain types of orientations on chordal graphs [43, 83]). Some problems that are NP-complete or #P-complete on general graphs can be solved in polynomial time on chordal graphs. For example, colorings and independent sets can be counted on chordal graphs in polynomial time [54,57]. Also , generating chordal graphs with a given number of vertices from the uniform distribution in polynomial time has been open for a long time, and it is crucial for testing the performances of some Bayesian network structural learning algorithms [25, 29, 30, 71].

1.1 Studied Graph Structures and Motivations

In this thesis, we studied counting and sampling of the following graph structures on chordal graphs. All these structures are widely studied by the theoretical community, and some of them are also important research topics in areas like statistical physics and probabilistic graphical models. Following prior lines of work, aiming to understand whether these structures can be counted and/or sampled in polynomial time on specific graph classes, we focus on chordal graphs, where some prior success has been demonstrated. It has been proved that the two classical #P-complete problems, counting of acyclic orientations and counting of independent sets, can be solved on chordal graphs

in polynomial time, obtaining an exact count. Our research focuses on counting and sampling of additional well-known graph structures on chordal graphs, and furthermore, on the efficient calculation of the Tutte polynomial for certain inputs on chordal graphs. The following are the graph structures studied in this thesis:

- Independent sets. Recall that an independent set of a graph is a subset of its vertices such that no two of them are connected by an edge. Sampling of weighted independent sets for a given graph G and a so called fugacity parameter $\lambda \in \mathbb{R}^+$ aims to produce an independent set S of G with probability proportional to $\lambda^{|S|}$, where $|S|$ is the number of vertices in S (when $\lambda = 1$, the goal is to sample independent sets uniformly at random). This problem is known as the hard-core model in statistical physics. For general graphs and arbitrary λ, the counting version of the problem is #P-complete. However, after a long line of research, some FPRASs were designed for graph classes with certain upper bounds on the vertex degrees and a range of λs dependent on the maximum degree [4,16,19,47,80]. The community has also studied this problem on different underlying graphs, and similar results about polynomial sampling have been shown on graphs with bounded bipartite pathwidth [12], bounded treewidth [11], and bounded connective constant [64]. Study of counting and sampling of independent sets is also crucial for the classification of the class #P from the approximate counting perspective [18]. For instance, counting of independent sets on bipartite graphs is believed to not have an FPRAS, but is not as hard as #SAT, the problem of counting assignments to variables that satisfy a given formula.

  We note that the problem of counting all independent sets is self-reducible, that is, roughly speaking, the problem can be solved using smaller instance of the same problem. [1] Recall that self-reducibility is a property that can be used to derive (almost) uniform samplers from counting algorithms (and vice versa, approximate counting algorithms from sampling algorithms).) Therefore, there are also corresponding sampling algorithms for independent sets for all the graph classes for which an FPRAS exists.

- Acyclic orientations. An acyclic orientation of an undirected graph is an assignment of edge directions that makes the graph directed and acyclic. Counting of acyclic orientations is a special case of the Tutte polynomial, and it has been proved to be #P-complete on general graphs [46]. Interestingly, the counting of acyclic orientations on chordal graphs can be done in polynomial time because it is closely related to the chromatic polynomial, which can be

[1] In particular, the set of all independent sets of G can be split into two subsets: those that do not include a vertex v (and are therefore an instance of the same problem for $G - v$), and those that do, where the remaining part of the independent set is in G' defined as G except v and its neighbors (an instance of the same problem in G').

computed in polynomial time on chordal graphs [1,67]. But, to the best of our knowledge, the problem of counting acyclic orientations is not known to be self-reducible, and no polynomial-time uniform sampler was known before the work presented in the thesis.

- Bipolar orientations . A bipolar orientation of an undirected graph is an assignment of a direction to each edge that makes the graph directed, acyclic, and with a single source s and a single sink t. Bipolar (s,t)-orientations, also known as st-numberings, are a natural restriction of acyclic orientations with applications in, for example, planarity testing [45]. While finding a bipolar orientation can be done in linear time by depth-first search [72], to the best of our knowledge no efficient algorithm for counting or sampling of bipolar orientations is known, even for restricted graph classes. The exact counting problem is #P-complete on general graphs due to a simple reduction from counting acyclic orientations.

- Sink-free orientations. An orientation is sink-free if it does not have any vertex with out-degree zero. Bubley and Dyer [15] proved that counting of sink-free orientations is #P-complete, and researchers have designed polynomial-times samplers for sink-free orientations based on different techniques [15, 17, 31].

- Source-sink-free orientations. We introduce a source-sink-free orientation as a natural restriction of a sink-free orientation, where the directed graph has neither vertex with indegree zero nor vertex with outdegree zero. A source-sink-free orientation is also a "weaker" version of a strong orientation (which corresponds to $T_G(0,2)$ [44]), which is an assignment of edge directions that makes the underlying graph strongly connected (each vertex can be reached from any other vertex). It is straightforward that every strong orientation is source-sink-free, but every source-sink-free orientation is not necessarily strongly connected. Counting of strong orientations of a graph is #P-complete, even when the graph is planar and bipartite [78, 81].

- Partial acyclic orientations. For an undirected graph, a partial acyclic orientation is an assignment of directions to a subset of its edges such that the directed edges do not form a directed cycle. Studying these types of orientations could shed light on the problem of sampling acyclic orientations (possibly also in general graphs), by assigning partial orientations with undirected edges a significantly smaller proportion in the overall distribution.

- Random chordal graphs. For a positive integer n, consider the set of all chordal graphs with n vertices (possibly also satisfying other desired properties like connectivity or an upper-bound on the treewidth). Our goal is to generate a uniformly random chordal graph from this set, which is necessary for testing the performance of some structural learning algorithms of Bayesian networks [22, 55, 60]. To the best of our knowledge, existing algorithms generating

chordal graphs sample chordal graph instances which do not have guaranteed distributions [49, 61, 62]. This means that testing an algorithm's performance on such samples might give a false sense of security since it is performed only on graph instances that are likely to be generated.

1.2 Contributions and Organization of the Thesis

In Chapter 3, we prove that a well-known Dyer&Greenhill Markov chain [19] can sample weighted independent sets for arbitrary λ in polynomial time on chordal graphs with a bound on minimal separator size. Our Markov chain result was published at COCOON '20 [8].

In Section 4.2, we present the first known uniform sampler of acyclic orientations in chordal graphs, which can provide a sample in time linear in the number of edges without any preprocessing. This work was published in AAAI '22 [68] and WALCOM '22 [9].

In Section 4.3, we introduce an efficient polynomial-time counter and sampler for bipolar orientations in chordal graphs. This work was published in WALCOM '22 [9].

In Section 4.4, we present a polynomial-time exact counter for sink-free orientations in chordal graphs. Our approach significantly improves the running time over the FPRAS for chordal graphs. This work was published in WALCOM '22 [9].

In Section 4.5, we present our initiation of research in counting the novel source-sink-free orientations. Interestingly, previous techniques that apply to sink-free orientations on general graphs do not appear to be applicable to the problem of source-sink-free orientations, which is our main motivation for studying them on chordal graphs. We extended our counting algorithm for sink-free orientations, using a more complex inclusion-exclusion principle (essentially, a sequence of additions and subtractions that account for over/under-counting) that adds a linear factor to the running time, to obtain a still reasonably efficient counting algorithm for source-sink-free orientations on chordal graphs. This work was published in WALCOM '22 [9]. .

In Section 4.6, we investigate the applicability of the partial rejection sampling framework [27] on the problem of sampling partial acyclic orientations from a clearly specified and natural (Gibbs-like) underlying distribution. This work was published in AAAI '21 [70].

In Chapter 5 we present a Markov chain to sample connected chordal graphs with a given number

of vertices and a bound on the treewidth. We prove that this chain, when restricted to chordal graphs of treewidth one and a given number of so-called simplicial vertices, can generate a sample in polynomial time. In other words, the produced chordal graph is a random tree with a prescribed number of leaves. This work was published in AAAI '20 [69].

Chapter 2

Preliminaries

In this chapter, we first introduce some basics of graph theory that are essential for understanding the thesis. Then we present the concept of approximate counting, almost uniform sampling, and how to convert them to each other when the underlying problem satisfies the self-reducible property. Next, we briefly talk about the Tutte polynomial, which is an important research topic from the counting perspective and is also related to multiple graph structures we studied in this thesis. Finally, we provide a brief introduction to the Markov chain Monte Carlo methods, a technique that has been widely used for statistical inference and sampling of graph structures.

2.1 Basics of Graph Theory

2.1.1 Graph structures

In this section, we will introduce some basic concepts in graph theory and graph structures that we studied, which include chordal graphs, independent sets, and graph orientations.

A graph G is an ordered pair $G = (V(G), E(G))$, where $V(G)$ is the set of vertices in G and $E(G) \subseteq \{(x, y) \mid (x, y) \in V(G)^2 \text{ and } x \neq y\}$ is the set of edges in G. When clear from the context, we use V and E in place of $V(G)$ and $E(G)$. A graph is undirected if all its edges are unordered pairs, and it is directed if all its edges are ordered pairs. An undirected graph G is *connected* if any two vertices (x, y) in the graph are connected by a *path* $(x = v_1, \ldots, v_l = y)$ such that $(v_i, v_{i+1}) \in E(G)$ for $i = 1, \ldots, l-1$. In the rest of the thesis, underlying graphs are always

undirected and connected. For a graph G, we denote by $G[U]$ the induced subgraph in G on the vertex set $U \subseteq V(G)$, where the edge set $E(G[U])$ contains all edges in $E(G)$ that have both endpoints in U. A *connected component* of a graph is a subgraph that is connected and is not part of any larger connected subgraphs. An *independent set* is a set of vertices in a graph where no two of these vertices are connected by an edge. An independent set is said to be maximal if it is not a proper subset of any other independent set. A *clique* in a graph is a subset of vertices such that every pair of vertices is connected by an edge. And a clique is maximal if it is not a subset of any clique of a larger size. In an undirected graph G, a *separator* is a vertex subset $S \subseteq V(G)$ whose removal from G disconnects some pair of vertices that were previously connected in G (i.e., there was a path between them in G). A separator is *minimal* if no proper subset of it is a separator. A tree T is a connected undirected graph that has a vertex set $V(T)$ and an edge set $E(T)$ of size $|V(T)| - 1$. Similar to trees, a *forest* is a set of disjoint trees, i.e., no two trees share common vertices.

An *orientation* of an undirected graph G is an assignment of a direction to each edge in G: an undirected edge $e = (u, v)$ becomes either the directed edge (u, v) or the directed edge (v, u). A *cycle* in a directed graph is a path $(v_1, \ldots, v_l)$ such that $l \leq 2$ and $v_1 = v_l$. For undirected graphs, we define a *(simple) cycle* as a path $(v_1, \ldots, v_l)$ such that $l > 2$, $v_1 = v_l$, and only vertex v_1 is repeated on the path. Let $\vec{G}$ be the directed graph corresponding to an orientation of G. This orientation is (1) *acyclic* if $\vec{G}$ is acyclic, (2) *bipolar* if $\vec{G}$ is acyclic and it contains a unique vertex of indegree 0 (a *source*) and a unique vertex of outdegree 0 (a *sink*), (3) *sink-free* if $\vec{G}$ contains no sinks, and (4) *source-sink-free* if $\vec{G}$ contains no sources and no sinks.

2.1.2 Chordal graphs and clique tree

An undirected graph is *chordal* if for every cycle of more than three vertices there exists an edge, called a chord, not on this cycle connecting two vertices on the cycle. In a given graph, a vertex is *simplicial* if all its neighbors form a clique (vertex A in Figure 2.1 is simplicial because its neighbors B and C form a clique). After removing a simplicial vertex from a chordal graph, the resulting graph is still chordal. A *perfect elimination order* is an ordering of the vertices such that each vertex in the ordering is simplicial with regard to the neighbors after it. For the chordal graph in Figure 2.1, ADCHEFGB is a perfect elimination order. To see this, we first remove vertex A and its incident edges, the resulting graph is chordal. Then we remove vertex D and its incident edges, which still gives us a chordal graph. We can keep removing the remaining vertices until we get an empty graph without breaking the chordality during this process. Rose, Leuker and Tarjan [59]

showed that a graph has a perfect elimination order if and only if it is chordal. They also provided a linear-time algorithm to find a perfect elimination order on chordal graphs.

Every chordal graph G can be represented by a tree T_G where $V(T_G)$ is the set of maximal cliques of G, and the tree satisfies the *induced subtree property*: For every vertex $v \in V(G)$, the induced subgraph $T_G[A_v]$ is connected, where A_v is the set of maximal cliques of G containing v. Such a tree T_G is called a *clique tree* of G (an example is shown in Figure 2.1), see, for example, [75] . The *treewidth* of a chordal graph equals to the size of its largest maximal clique minus one (The treewidth is defined for general graphs as well, but we do not need the definition in this work.). Treewidth is usually used as a parameter in parameterized complexity analysis of algorithms, e.g., design and analysis of algorithms on graphs with constant treewidth. Let T_{G,C_r} be the clique tree T_G rooted at a maximal clique C_r. If G is clear from the context, we will simply write T_{C_r}, or simply T if C_r is also clear. We denote by $T_{C_r,C}$ the subtree of T_{C_r} containing C and its descendants; we write T_C if C_r is clear from the context. Each clique C in T_{C_r} can be partitioned into a *separator set* $\text{Sep}(C) = C \cap \text{Parent}(C)$ and a *residual set* $\text{Res}(C) = C \setminus \text{Sep}(C)$, where $\text{Parent}(C)$ is the parent clique of C in T_{C_r} (if $C = C_r$, then $\text{Parent}(C) = \emptyset$). The following properties hold, see, for example, [10, 75] (see Figure 2.1):

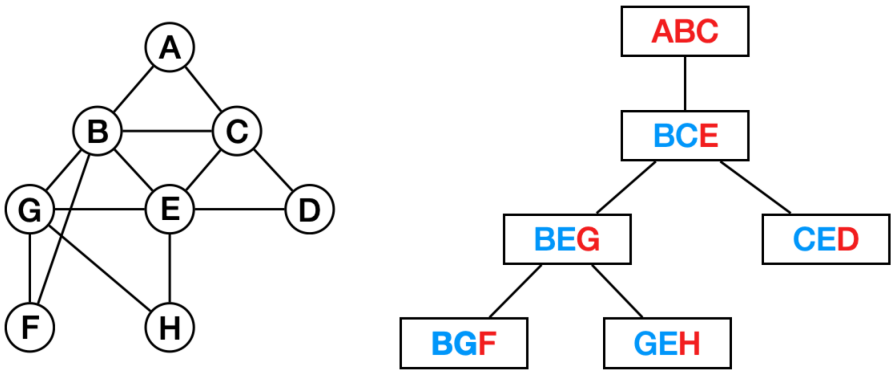

Figure 2.1: A chordal graph G and one of its rooted clique trees T_{ABC}. Vertices in separator sets are in blue, and vertices in residual sets are in red.

- For each vertex v in G, there is a unique clique C_v that contains v in its residual set. This implies that $|V(T_G)| \leq |V(G)|$ and that C_v is the root of $T_{C_r}[A_v]$; we denote this rooted subtree by T_{C_v}. All other cliques in T_{C_v} that contain v have it in their separator set.
- For a clique C let $D(C)$ be the set of vertices in the descendant cliques of C in T_{C_r} except $\text{Sep}(C)$, i.e., $D(C) := \bigcup_{C' \in V(T_C)} C' - \text{Sep}(C)$. Let $A(C)$ be the vertices in the cliques not

in T_C except Sep(C), i.e., $A(C) := \bigcup_{C' \in V(T_{C_r}) - V(T_C)} C' - \text{Sep}(C)$. The separator Sep($C$) separates $A(C)$ and $D(C)$ in G: there is no edge with one endpoint in $A(C)$ and the other endpoint in $D(C)$.

- Construction of a clique tree for a connected chordal graph can be done in time $O(|E(G)|)$.

2.2 Counting and Sampling

2.2.1 Relationship between counting and sampling

Given a set Ω and a weight function $w : \Omega \to \mathbb{R}^+$, the goal of weighted counting is to calculate the sum of the weights of members in Ω, i.e., $\sum_{x \in \Omega} w(x)$, which we call the *partition function* with respect to Ω and w, typically denoted by $Z_{\Omega,w}$, or just Z when Ω and w are clear from the context. Given an error parameter $0 < \epsilon < 1$ and a probability of failure p_{fail} ($p_{\text{fail}} < 1/2$), a fully polynomial-time randomized approximation scheme (FPRAS) is a randomized algorithm that outputs a number $\tilde{Z}$ such that

$$P\left((1-\epsilon)Z \leq \tilde{Z} \leq (1+\epsilon)Z\right) \geq 1 - p_{\text{fail}},$$

and the running time of the algorithm is polynomial in the input size (for graph problems, the input size can be viewed as the number of vertices and edges in the input graph), $\log p_{\text{fail}}^{-1}$ and ϵ^{-1}. For a given set Ω, a weight function w, and an error parameter δ, a fully polynomial time almost uniform sampler (FPAUS) is an algorithm that outputs elements from Ω according to a distribution μ such that

$$\sum_{x \in \Omega} \left| \mu(x) - \frac{w(x)}{Z} \right| \leq \delta,$$

and its running time is polynomial in the input size and $\log \delta^{-1}$. For self-reducible problems, FPAUS and FPRAS are equivalent, i.e., we can use one to construct the other. A counting problem is *self-reducible* if the set of solutions can be partitioned into polynomially many sets, and the solutions in each of these sets have a one-to-one correspondence with the solutions of a smaller instance of the problem.

[1]

[1] AFTER SHOWING SELF-REDUCIBLITY OF MATCHINGS, SHOW HOW TO GET AN FPAUS FROM AN FPRAS - OR FROM AN EXACT COUNTING ALGORITHM. THEN FOLLOW WITH THE FPRAS USING FPAUS.

Here we use an example of matchings to show how to construct an FPRAS using an FPAUS and vice versa . Given a graph $G = (V, E)$, a matching M is a subset of the edge set E such that no two edges in M have common vertices. We first label edges of G by $e_1, \ldots, e_m$, then we define a graph sequence of subgraphs of G: $G_0, \ldots, G_m$ where $G_0 = G$ and $G_i = G_{i-1} - e_i$ for $1 \leq i \leq m$. Let $\Omega(G_i)$ be the set of matchings in G_i, then the number of matchings in $\Omega(G)$ is $|\Omega(G)|$ and it can be expressed by

$$|\Omega(G)| = \left(\frac{|\Omega(G_1)|}{|\Omega(G_0)|} \cdot \frac{|\Omega(G_2)|}{|\Omega(G_1)|} \cdots \frac{|\Omega(G_m)|}{|\Omega(G_{m-1})|}\right)^{-1} = \frac{|\Omega(G_0)|}{|\Omega(G_m)|} = |\Omega(G)|,$$

where the last equality holds because $|\Omega(G_m)| = 1$ since G_m contains no edges. Let p_i denote the ratio $|\Omega(G_i)|/|\Omega(G_{i-1})|$, then estimating each p_i with sufficiently small error will give us an estimate of $|\Omega(G)|$. Also, we need to notice that the error upper bound of p_i depends on the given error upper bound of $|\Omega(G)|$ (δ depends on ϵ). Then we need to build a sampler over each $\Omega(G_{i-1})$ such that it can get a sample that is in $\Omega(G_i)$ with a probability that is close enough to p_i. To get an FPRAS, the sampling procedure needs to satisfy the following requirements:

1. The number of subproblems should be polynomially bounded. For counting matchings, we have m subproblems, i.e., estimating the m ratios.
2. The ratio $p_i = \frac{|\Omega(G_i)|}{|\Omega(G_{i-1})|}$ should be polynomially small in the input size, otherwise we might need superpolynomially (possibly exponentially) many samples to get one sample that falls into $\Omega(G_i)$. To bound p_i for the matching problem, we can divide matchings in $\Omega(G_{i-1})$ into two sets S_1 and S_2, where S_1 contains matchings that have e_i and S_2 contains matchings that do not have e_i. Obviously, $S_2 \subseteq \Omega(G_i)$, and for each matching M in S_1, $M - e_i$ also belongs to $\Omega(G_i)$. Therefore, $p_i \geq 1/2$.
3. There is an FPAUS for sampling for each subproblem. For the matching problem, we need to be able to sample matchings almost uniformly at random on an input graph in polynomial time.

For the matching problem, to convert an FPAUS to an FPRAS, we need sufficiently many samples to estimate each ratio p_i, from which we can get an estimate of $|\Omega(G)|$. A lower-bound on p_i provides a bound on the number of samples needed to estimate the expected value.

To convert an FPRAS to an FPAUS, for the current graph $G = (V, E)$, we can choose an edge $e = (u, v) \in E$ uniformly at random and use it to split $\Omega(G)$ into two exclusive sets $\Omega(G_1)$ and $\Omega(G_2)$ where $G_1 = (V, E \backslash e)$ and $G_2 = G[V \backslash \{u, v\}]$. The probability that e should be included in

the matching that is currently being generated can be computed from $|\Omega(G_1)|$ and $|\Omega(G_2)|$: namely, include edge e in the current matching with probability $|\Omega(G_2)|/|\Omega(G)| = |\Omega(G_1)|/(|\Omega(G_1)+\Omega(G_2)|$, otherwise, omit it. If e is included, proceed to generate other edges in the matching using the graph G_2, otherwise, use G_1. This recursive process gives us a uniformly random matching if we have access to an exact counting algorithm, and an almost uniformly random matching if we use an FPRAS.

This high-level exposition is sufficient for the purposes of this thesis, for more details, for example the calculation of the required number of samples, we refer the reader to [33].

2.2.2 Tutte polynomial

The Tutte polynomial relates to the counts of several graph structures that we studied in this thesis, such as acyclic orientations and independent sets. In this section, we briefly introduce the basics of the Tutte polynomial.

The Tutte polynomial $T_G(x, y)$ of an undirected graph $G = (V, E)$ with two parameters x, y is defined as

$$T_G(x,y) = \sum_{A \subseteq E} (x-1)^{k(A)-k(E)}(y-1)^{k(A)+|A|-|V|}, \tag{2.1}$$

where $k(A)$ is the number of connected components in the subgraph (V, A). Let us see an example, suppose the $G = K_3$ (the complete graph with three vertices):

$$\begin{aligned} T_{K_3}(x,y) =& \sum_{A \subseteq E} (x-1)^{k(A)-k(E)}(y-1)^{k(A)+|A|-|V|} \\ =& \underbrace{(x-1)^{1-1}(y-1)^{1+3-3}}_{\{e_1,e_2,e_3\}} + 3\underbrace{(x-1)^{1-1}(y-1)^{1+2-3}}_{\{e_1,e_2\},\{e_1,e_3\},\{e_2,e_3\}} \\ &+ 3\underbrace{(x-1)^{2-1}(y-1)^{2+1-3}}_{\{e_1\},\{e_2\},\{e_3\}} + \underbrace{(x-1)^{3-1}(y-1)^{3+0-3}}_{\emptyset} \\ =& x^2 + x + y. \end{aligned}$$

Lots of important graph quantities are special cases of the Tutte polynomial. For example, the chromatic polynomial $P(G, k)$ counts the number of proper k-colorings in a graph G, where a proper k-coloring is an assignment of one of k colors to each vertex in G such that no two adjacent vertices share the same color. The number of acyclic orientations on a graph G, the $T_G(2, 0)$ [82], is related to the chromatic polynomial of G evaluated, quite surprisingly, at -1 [67]: in particular, it is $P(G, -1) \cdot (-1)^{|V(G)|}$.

Calculating $T_G(x, y)$ is generally hard. For example, Linial [46] showed that counting acyclic orientations is #P-complete (and, then, so is the chromatic polynomial). Jaeger, Vertigan, and Welsh [32] proved that calculating $T_G(x, y)$ is #P-hard, except along the hyperbola $H_1 = (x - 1)(y - 1) = 1$ and at the four special points $\{(1, 1), (0, -1), (-1, 0), (-1, -1)\}$. For convenience, we let H_q denote the hyperbola $(x - 1)(y - 1) = q$.

The Tutte polynomial is not only hard to calculate exactly, but it is also hard to calculate approximately. Goldberg and Jerrum [26] proved that FPRASs do not exist for most of the (x, y) pairs except for some special cases. Their results are shown in Figure 2.2 (the Figure 1 in [26]). From [26], we know that for a general graph G, there is no FPRAS for $P(G, x)$ for any $x > 2$, unless NP = RP. Here, RP is , roughly, a complexity class for problems with polynomial-time randomized algorithms; the exact definition is not needed for the purposes of this thesis. It should be noted that, similarly to the P vs. NP problem, where the general belief is that P is not equal NP, NP is likewise widely believed to not equal RP.

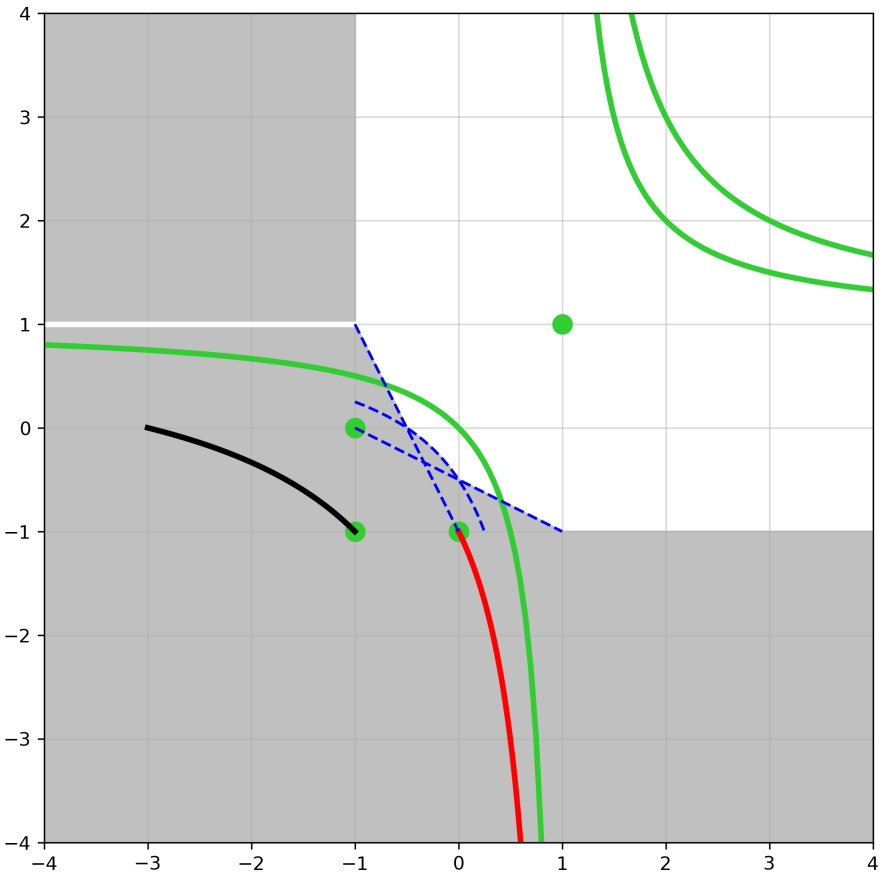

Figure 2.2: Illustrating approximability of the Tutte polynomial for general graphs: The four isolated green points and points on the green parts of the hyperbolas H_1 and H_2 are FPRASable. Red points are on H_2 with $y < -1$, and they are equivalent to counting of perfect matchings. A point is grey if its $x < -1$ or $y < -1$ and is not on any specific hyperbola, or it is in the area bounded by $|x| < 1$, $|y| < 1$, and the two dashed blue lines ($y < -1 - 2x$ and $x < -1 - 2y$) and the dashed blue curve ($(x-1)(y-1) > 1.5$). Grey points are not FPRASable unless RP=NP. The hardness of the white points (the white areas and the half-line $x < -1$ and $y = 1$) is unknown. The black points are on H_4 with $y \in (-1, 0)$, and they are at least as hard as grey points.

The two structures we studied in this thesis, independent sets and acyclic orientations, are special cases of the Tutte polynomial. Also, the source-sink-free orientations has a close relationship with another special case, strong orientations of a graph.

2.3 Markov chain Monte Carlo

The Markov chain Monte Carlo (MCMC) comprises a class of algorithms for sampling from a distribution over a state space. In this thesis, all state spaces are discrete and finite. This sampling approach is done by first constructing a Markov chain that has the desired target distribution (such as the uniform distribution) as its *stationary distribution* over the state space, then running this chain for "sufficiently many" steps to get a sample. Compared to constructing a Markov chain with the desired distribution, bounding the number of steps needed to get to the stationary distribution is usually much harder. In this section, we introduce the basics of Markov chains. For more details of MCMC, see for example [52].

2.3.1 Basics of Markov chains

A sequence $(X_t \in \Omega)_{t=0}^{\infty}$ of random variables is a Markov chain (MC) with state space Ω if

$$\Pr\left[X_{t+1} = y \mid X_t = x, \ldots, X_0 = x_0\right] = \Pr\left[X_{t+1} = y \mid X_t = x_t\right] = P(x, y) \tag{2.2}$$

for all $t \in \mathbb{N}$ and all $x_i \in \Omega$ for $0 \leq i \leq t$, where P is the transition matrix of the MC (an example of a Markov chain transition is shown in Figure 2.3). A Markov chain is usually represented by its state space Ω and the $|\Omega| \times |\Omega|$ transition matrix P, and the MC can also be represented by P alone. The transition matrix can be described explicitly, but often it is very sparse and is described by an algorithm. For an initial distribution μ over Ω (viewed as a row-vector with $|\Omega|$ entries), the distribution after t transitions is μP^t. A distribution π over Ω is a *stationary distribution* of the Markov chain P if $\pi P = \pi$; in other words, π is the long-term limiting distribution of the chain. A Markov chain P is *reversible* with regard to distribution π if $\pi(x)P(x, y) = \pi(y)P(y, x)$. If this condition is satisfied, then π is a stationary distribution of the chain P, which follows from

$$\sum_x \pi(x)P(x, y) = \sum_x \pi(y)P(y, x) = \pi(y) \sum_x P(y, x) = \pi(y).$$

A chain is *irreducible* if for any two states $x, y \in \Omega$, there exists an integer t such that $P^t(x, y) > 0$, which means any state can get to any other state in finite number of steps. Let $\mathcal{T}(x) := \left\{t \geq 1 : P^t(x, x) > 0\right\}$ be the set of times when it is possible for the chain to return to the starting state x; a chain is *aperiodic* if $\gcd \mathcal{T}(x) = 1$ for all $x \in \Omega$. If a chain is both irreducible and aperiodic, we call it *ergodic*.

Theorem 1. *An ergodic Markov chain P has a unique stationary distribution π such that*

$$\lim_{t \to \infty} \mu P^t = \pi$$

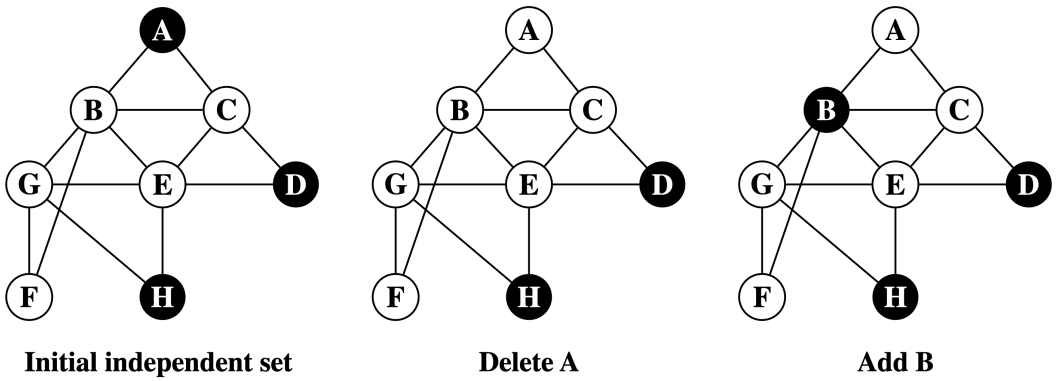

Figure 2.3: Illustrating a Markov chain for sampling independent sets of a given graph. The state space Ω is the set of all independent sets. A transition consists of choosing a uniformly random vertex: If it is in the current independent set, remove it. If not, and if it can be added, add it to the current independent set. The figure shows two transition steps of this Markov chain. The current independent set is shown in black. The Markov chain first picks vertex A and deletes it from the current independent set, then it picks vertex B, and adds it to the independent set. Each transition depends only on the current state.

for arbitrary initial distribution μ.

Given a state space Ω and a strictly positive weight function $w : \Omega \mapsto \mathbb{R}^+$, MCMC aims to construct a reversible, irreducible, and aperiodic Markov chain over Ω whose stationary distribution is $\pi(x) = \frac{w(x)}{Z}$ for $x \in \Omega$, where $Z = \sum_{x\in\Omega} w(x)$. Achieving a desired distribution over Ω is typically done by the Metropolis-Hasting algorithm [51], which has two components:

- Neighborhood structure: for two different elements $x, y \in \Omega$, they are neighbors if they can be converted to each other by some local changes. For instance, x and y can be two neighboring independent sets if they differ at one vertex. The neighborhood structure should ensure irreducibility.
- Proposal distribution: for each state x we have a transition distribution $p(x, \cdot)$ over Ω which satisfies the following properties:
  - $p(x, y) > 0$ if y is a neighbor of x.
  - $p(x, y) = p(y, x)$ for all $y \in \Omega$.
  - $\sum_{y\in\Omega} p(x, y) = 1$.

Once we have the above two components, to do a one-step transition of the Markov chain, we

1. For the current state x, pick a neighbor y with probability $p(x, y)$.
2. Move from x to y with probability $\min\left\{\frac{\pi(y)}{\pi(x)}, 1\right\}$.

Suppose $\pi(x) \leq \pi(y)$, the transition probability between x and y would be

$$P(x, y) = p(x, y) \cdot \min\left\{\frac{\pi(y)}{\pi(x)}, 1\right\} = p(x, y)$$
$$P(y, x) = p(y, x) \cdot \min\left\{\frac{\pi(x)}{\pi(y)}, 1\right\} = p(y, x) \cdot \frac{\pi(x)}{\pi(y)}.$$

Then, π is exactly the stationary distribution because $\pi(x)P(x, y) = \pi(y)P(y, x)$. To ensure the aperiodicity of the Markov chain P, we replace P with a new "lazy" chain $\tilde{P} := (P + I)/2$, where I is an identity matrix of the same size as P. It means that for any current state, $\tilde{P}$ has a non-zero probability to stay at that state.

2.3.2 Mixing time of Markov chains

The *total variation distance* (tvd) between two probability distributions μ and η on Ω is defined by

$$\|\mu - \eta\|_{\text{TV}} = \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \eta(x)| = \max_{A \subseteq \Omega} |\mu(A) - \eta(A)|$$

For an ergodic Markov chain P and its stationary distribution π, let μ_x be the initial distribution at state x (that is, $\mu_x(z) = 0$ for all $z \in \Omega$ such that $z \neq x$ and $\mu_x(x) = 1$), the total variation distance between the t-step distribution and the stationary distribution is defined as

$$d_x(t) := \max_{x \in \Omega} \left\|\mu_x P^t - \pi\right\|_{\text{TV}}$$

Given $0 < \epsilon < 1$, the mixing time $\tau_x(\epsilon)$ measures the time required by a Markov chain, when started from state x, to achieve the distance to stationarity smaller than ϵ:

$$\tau_x(\epsilon) := \min\{t : d_x(t) \leq \epsilon\}$$

We define the mixing time $\tau(\epsilon)$ as $\tau(\epsilon) := \max_x \tau_x(\epsilon)$, in other words, the mixing time is the time needed to get ϵ-close to stationarity from the "worst" possible start state.

Mixing time can be viewed as the number of steps a Markov chain needs to take to get one sample. Therefore, bounding the mixing time is the core part of MCMC algorithms.

2.3.3 Techniques for bounding the mixing time

In this section we will introduce a couple of techniques for bounding the mixing time of a Markov chain.

Coupling is a technique to bound the mixing time. A *Markovian coupling* of a Markov chain P is a Markov chain (X_t, Y_t) with state space $\Omega \times \Omega$ which satisfies for all $x, y, x', y' \in \Omega$:

$$\mathbf{P}\left\{X_{t+1} = x' \mid X_t = x, Y_t = y\right\} = P\left(x, x'\right)$$
$$\mathbf{P}\left\{Y_{t+1} = y' \mid X_t = x, Y_t = y\right\} = P\left(y, y'\right)$$

By the coupling lemma [2], given a total variation distance ϵ, if we can find a number $t(\epsilon)$, such that

$$\Pr\left[X_{t(\epsilon)} \neq Y_{t(\epsilon)} \mid X_0 = x, Y_0 = y\right] \leq \epsilon$$

then the mixing time is bounded by $t(\epsilon)$. This inequality is usually shown by first defining a distance metric d over $\Omega \times \Omega$ and showing the geometric decreasing of distance:

$$\mathrm{E}\left[d\left(X_{t+1}, Y_{t+1}\right) \mid X_t, Y_t\right] \leq (1-\alpha) d\left(X_t, Y_t\right)$$

Since it is not easy to define a distance metric d for arbitrary pairs of instances in Ω, Bubley and Dyer [15] introduced the *path coupling* technique. A pre-metric on Ω is a weighted connected graph on the vertex set Ω with positive edge weights with the property that every edge (u, v) is also the shortest path from u to v. We say that two elements in Ω are adjacent if there is an edge in the pre-metric. Once we have a pre-metric, it is sufficient to show the decrease of expected distance between adjacent pairs of states after taking one step of the coupled Markov chain. Concretely, the mixing time can be bounded by

$$\tau(\varepsilon) \leq \left\lceil \frac{1}{\alpha}[\log(\mathrm{diam}(\Omega)) + \log(1/\varepsilon)] \right\rceil, \tag{2.3}$$

where $\mathrm{diam}(\Omega)$ is the diameter of Ω, which is defined to be $\mathrm{diam}(\Omega) := \max_{x,y \in \Omega} d(x, y)$.

Although path coupling is very powerful, it is not always applicable for proving rapid mixing [6]. *Canonical paths* [34, 63] (see also [33] for a nice exposition) is another technique for bounding

the mixing time. The idea is to define , for every pair of states $x, y \in \Omega$, a canonical path $\gamma_{x,y} = (x = z_0, \ldots, z_\ell = y)$ such that (z_i, z_{i+1}) are adjacent states in the Markov chain, that is, $P(z_i, z_{i+1}) > 0$. Let $\Gamma := \{\gamma_{xy} \mid x, y \in \Omega\}$ be the set of all canonical paths. The *congestion* through a transition $t = (u, v)$, where $P(u, v) > 0$, is

$$\varrho(\Gamma, t) := \frac{1}{\pi(u)P(u,v)} \sum_{x,y:\gamma_{xy} \text{ uses } t} \pi(x)\pi(y) \left|\gamma_{xy}\right| \tag{2.4}$$

where $|\gamma_{xy}|$ is the length of the path γ_{xy}. Intuitively, a Markov chain mixes fast when none of its transitions is overloaded. This can be seen from two perspectives:

- First is that $\pi(u)P(u, v)$ should not be too small. This quantity is the capacity of t, which can be viewed as the narrowness of a highway between two cities.
- Second is $\sum_{x,y:\gamma_{xy} \text{ uses } t} \pi(x)\pi(y) \left|\gamma_{xy}\right|$, which is the total probability flow through t. This can be viewed as the rate of traffic flow over a highway between two cities.

The overall congestion of the paths Γ is defined as

$$\varrho(\Gamma) := \max_{t=(u,v):P(u,v)>0} \varrho(\Gamma, t). \tag{2.5}$$

The mixing time of the lazy chain is bounded by

$$\tau_x(\epsilon) \leq 2\varrho(\Gamma) \left(\ln(\pi(x)^{-1}) + 2\ln(\epsilon^{-1})\right). \tag{2.6}$$

Typically, $\ln \pi(x)^{-1}$ is in poly(n), where n is the input size [2], so our target is to bound the congestion. This can be achieved by the "flow encoding" technique.

Definition 1. *Let $t = (u, v)$ be a transition and let $\tilde{\Gamma}_t$ be the set of pairs (x, y) for which the canonical path γ_{xy} uses t. A flow* encoding *of $\tilde{\Gamma}_t$ is a mapping $\eta_t : \tilde{\Gamma}_t \to \Omega \times B$ for some set B, such that*

- *η_t is injective. This assures that from $\eta_t(x, y)$ we can uniquely recover (x, y).*
- *$\pi(x)\pi(y) \leq \beta\pi(u)\pi(\eta_t(x, y)_1)$ for some β, where $\eta_t(x, y)_1$ is the first component (the one in Ω) of $\eta_t(x, y)$.*

[2] more explanation?

Once we have an encoding, the congestion in Equation (2.4) can be bounded by

$$\begin{aligned}
\varrho(\Gamma, t) &= \frac{1}{\pi(u)P(u,v)} \sum_{x,y:\gamma_{xy} \text{ uses } t} \pi(x)\pi(y)\,|\gamma_{xy}| \\
&\leq \frac{1}{\pi(u)P(u,v)} \sum_{x,y:\gamma_{xy} \text{ uses } t} \beta\pi(u)\pi(\eta_t(x,y)_1)\,|\gamma_{xy}| \\
&\leq \frac{\max(|\gamma_{xy}|)}{\pi(u)P(u,v)} \sum_{x,y:\gamma_{xy} \text{ uses } t} \beta\pi(u)\pi(\eta_t(x,y)_1) \\
&= \frac{\max(|\gamma_{xy}|)}{\pi(u)P(u,v)} \sum_{(z,b)\in\Omega\times B:\ \exists(x,y)\in\tilde{\Gamma}_t:\ \eta_t(x,y)=(z,b)} \beta\pi(u)\pi(z) \\
&\leq \frac{\beta\max(|\gamma_{xy}|)}{P(u,v)} \sum_{(z,b)\in\Omega\times B} \pi(z) \\
&= \frac{\beta\max(|\gamma_{xy}|)}{P(u,v)} \sum_{b\in B}\sum_{z\in\Omega} \pi(z) \\
&= \frac{\beta|B|\max(|\gamma_{xy}|)}{P(u,v)}.
\end{aligned}$$

If $\max(|\gamma_{xy}|)$, $P(u,v)^{-1}$, $|B|$, and β are polynomial in the input size, we get a polynomial bound on the congestion. An example of using this technique to bound the mixing time will be shown in the following chapter.

Chapter 3

Sampling Independent Sets

In this chapter we present the first result of this thesis, showing that a well-known Dyer&Greenhill Markov chain [19] can sample weighted independent sets in polynomial time on chordal graphs with a bound on minimal separator size. This result was published at COCOON '20 [8].

Independent sets are heavily studied in computer science and other fields. Among their many applications is the hardcore model of gas in statistical physics, where the goal is to sample independent sets of a given graph according to a specific probability distribution [82] . In particular, for a given parameter (also known as fugacity) $\lambda \in \mathbb{R}^+$, the goal is to generate an independent set S with probability proportional to $\lambda^{|S|}$, where $|S|$ is the number of vertices in S. Thus, the probability of a set S is $\lambda^{|S|}/Z_G(\lambda)$, where the normalization term (also referred to as the partition function) $Z_G(\lambda)$ is defined as the sum of $\lambda^{|S|}$ across all independent sets S of G. The distribution favors small independent sets for $\lambda < 0$, large independent sets for $\lambda > 1$, and it samples uniformly at random for $\lambda = 1$.

In this chapter, we assume that for a given chordal graph G, there exists a constant $b \in \mathbb{N}^+$, which we refer to as the *separator bound*, that upper-bounds the size of every minimal separator of G. We study a well-known Markov chain, the mixing time of which is unknown for general graphs. The assumption on the separator size allows us to make progress with the understanding of the mixing time of this Markov chain on a class of graphs for which polynomial mixing time was unknown prior to our work. As a side remark, we note that for a given graph G and a constant b, one can check, in linear time, whether G is chordal and has a separator bound b (any two maximal cliques share at most b vertices). We obtain a mixing time of $O(n^{O(\log b)})$ for the well-known Dyer&Greenhill Markov chain [19] for arbitrary λ and chordal graphs with minimal separators of size at most b (or,

equivalently, bound b on the intersection size of any pair of maximal cliques).

3.1 Related Work

Markov chains have attracted attention as a sampling technique for the hardcore distribution since the late 1990s. The mixing time of a Markov chain is the time it takes to converge to its stationary distribution, and a Markov chain is said to be rapidly mixing if its mixing time is polynomial in the size of the input. Luby and Vigoda [47] showed rapid mixing of a natural insert/delete chain (a single-site Glauber dynamics) for independent sets of triangle-free graphs with degree bound Δ and $\lambda < 2/(\Delta - 2)$, which is soon extended by Vigoda [79] to general graphs with degree bound Δ. Independently, Dyer and Greenhill [19] analyzed an insert/delete chain with an added drag transition, showing rapid mixing for the same graph class and range of λs. All these works used the coupling technique to obtain their mixing results.

Weitz [80] proposed a *correlation decay* method that can estimate the partition function in polynomial-time when $\lambda < \lambda_c(\Delta) := (\Delta-1)^{\Delta-1}/(\Delta-2)^{\Delta}$, where $\lambda_c(\Delta)$ is called the uniqueness threshold on the Δ-regular tree. This method can generate an independent set uniformly at random in polynoial when $\Delta \leq 5$. A hardness result of Dyer, Frieze, and Jerrum [21] followed, showing that even for $\lambda = 1$, no Markov chain for sampling independent sets that changes only a "small" number (that is, a linear fraction) of vertices per step mixes rapidly for general graphs, even if the maximum degree is upper-bounded by six. Later on, Sly [66] proved that for any $\Delta \geq 3$, there exists a positive constant $\epsilon(\Delta)$ such that when $\lambda_c(\Delta) < \lambda < \lambda_c(\Delta) + \epsilon(\Delta)$, there is no polynomial-time approximation algorithm for estimating the partition function $Z_G(\lambda)$, unless NP=RP. This implies that unless RP=NP, no fully polynomial approximation scheme exists for counting independent sets on graphs of maximum degree at most Δ. As a further step, Galanis, Ge, Štefankovič, Vigoda, and Yang [23] removed the constraint of $\epsilon(\Delta)$ and improved Sly's bound to $\lambda < \lambda_c(\Delta)$. Anari, Liu, and Gharan [4] provided a similar result to [23] which shows that for every $0 < \delta < 1$ and $\lambda < (1-\delta)\lambda_c(\Delta)$, the single-site Glauber dynamics for the hard-core model mixes rapidly on any graphs with maximum degree less than or equal to Δ, and, therefore, there exists an FPRAS for estimating the partition function of the hard-core model in this domain. This result was recently further improved to $O(n \log n)$ mixing time by Chen, Liu, and Vigoda [16].

The research community (both theoretical computer science and statistical physics) is also interested in sampling and counting independent sets on special types of graphs. Theoretical computer science is interested in learning the boundaries (graph classes, parameter settings) for which prob-

lems can be solved in polynomial time, while statistical physics encounters special graph classes in their applications. The *connective constant* is, roughly, a measure of the average degree of a graph, so it can be bounded by a constant even when the underlying graph has no maximum degree bound. Sinclair, Srivastava, Štefankovič, an Yin [64] designed an FPTAS (and FPTAS is defined analogously to an FPRAS, but it is deterministic, with no associated error probability) for graphs of connective constant Δ whenever the vertex activity $\lambda < \lambda_c(\Delta)$, where $\lambda_c(\Delta) := \frac{\Delta^\Delta}{(\Delta-1)^{\Delta+1}}$. Bordewich and Kang [12] studied an insert/delete Markov chain (a multi-site Glauber dynamics) to sample vertex subsets, a generalization of the hardcore model, and proved that its mixing time is $n^{O(\mathrm{tw})}$ for an arbitrary λ and n-vertex graphs of treewidth tw. Very recently, generalizing work of Matthews [50] on claw-free graphs, Dyer, Greenhill, and Müller [20] introduced a new graph parameter, the bipartite pathwidth, obtaining a mixing time of $n^{O(p)}$ for the insert/delete chain for an arbitrary λ and graphs with bipartite pathwidth bounded by p. These works used the canonical paths technique [34] to prove rapid mixing. Interestingly, Okamoto, Uno, and Uehara [56] designed polynomial-time dynamic programming algorithms to count independent sets, maximum independent sets, and independent sets of fixed size on chordal graphs without any restrictions, which can then be used to sample independent sets from the hardcore distribution.

3.1.1 Dyer&Greenhill chain

Recall that we are given a graph G and a parameter $\lambda \in \mathbb{R}^+$. The most commonly used Markov chain for sampling independent sets is the Glauber dynamics (also known as the Luby-Vigoda chain or the insert/delete chain): Let S be the current independent set. Pick a random vertex $u \in V(G)$. If $u \in S$, remove it from S with a probability dependent on λ to maintain the desired target distribution. Namely, in the target distribution the probability of the set S is $\lambda^{|S|}/Z_G(\lambda)$, which can be achieved by doing the "remove transition" with probability $\frac{1}{1+\lambda}$. If $u \notin S$ and if none of its neighbors are in S, add it to S with probability $\frac{\lambda}{1+\lambda}$. Our polynomial mixing time results hold for the Glauber dynamics but in this work, we prove mixing time bounds for a closely related Markov chain by Dyer and Greenhill [19].

The Dyer&Greenhill chain: Let $S \in \Omega_G$ be the current independent set. Pick a vertex u uniformly at random from $V(G)$. Then:

[Delete $\downarrow$:] If $u \in S$, remove it with probability $1/(1+\lambda)$.

[Insert $\uparrow$:] If $u \notin S$ and none of the neighbors of u are in S, add u with probability $\lambda/(1+\lambda)$.

[Drag $\leftrightarrow$:] if $u \notin S$ and it has a unique neighbor $v \in S$, add u and remove v with probability $\lambda/(4(1+\lambda))$.

Otherwise, do nothing.

Let S' be the resulting independent set (if none of the above holds for u, let $S' = S$), which is the next state of the Markov chain. This chain is a variant of the Metropolis filter. It is ergodic with the desired stationary distribution $\pi(S) = \frac{\lambda^{|S|}}{Z_G(\lambda)}$ [19].

3.2 Our Contribution

We use the canonical paths technique but we need to overcome the "non-linearity" of our data. The technique relies on finding a Markov chain path (a canonical path) between every pair of states in such a way that no transition gets overloaded (congested). This is typically done by considering the symmetric difference of the two states (a pair of independent sets), gradually removing one vertex from the initial independent set while adding a vertex from the final independent set. If the symmetric difference induces a collection of paths in the original graph, we can "switch" each path from initial to final starting at one end-point of the path and gradually going to the other end-point, never violating the independent set property. Bounded bipartite pathwidth guarantees that the symmetric difference can be viewed as "wider" paths, as does bounded treewidth due to its relation to the (non-bipartite) pathwidth. However, for our graphs, the symmetric difference is tree-like, which leads to the need to recursively "switch" entire subtrees from initial to final before being able to process the root vertex from the final independent set. Due to this "tree-like" process, we also need to overcome corresponding complications in the analysis of the congestion.

3.2.1 Canonical paths for chordal graphs

From now on we assume that G is a connected chordal graph with n vertices. We will define a canonical path between every pair of independent sets I ("initial") and F ("final") in G. As is often done in canonical paths construction, we will work only with vertices of $I \oplus F$, the symmetric difference of I and F: we will gradually remove vertices from $I \setminus F$ while adding vertices in $F \setminus I$. (Notice that vertices in $I \cap F$ do not neighbor $I \oplus F$, and hence we do not need to touch them.)

We first observe that the symmetric difference of two independent sets in a chordal graph forms an induced forest (see Figure 3.1).

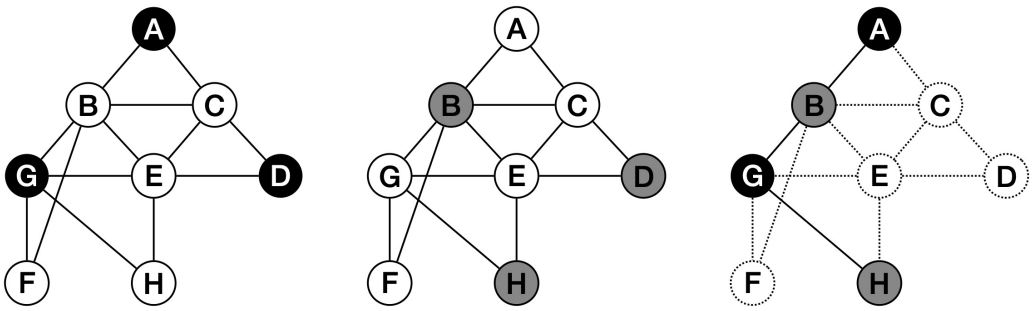

Figure 3.1: The black vertices in the left graph form an independent set I. The grey vertices in the middle graph form an independent set F. The induced subtree on the right is their symmetric difference (dashed vertices and edges are not in the symmetric difference).

Lemma 2. *Let G be a chordal graph and let I and F be its two independent sets. Then, the subgraph of G induced by $I \oplus F$ is a forest.*

Proof. Let H be the subgraph induced by $I \oplus F$. By contradiction, assume that H contains a cycle D. Since G is chordal and H is induced, D must have a chord in H, obtaining a shorter cycle. Applying this argument inductively, we get that H contains a triangle T. But then T must either contain two I-vertices or two F-vertices, which is impossible because both I and F are independent sets. Therefore, H is a forest. □

We assume that the vertices of G are labeled $1, \ldots, n$. Before defining our canonical paths, we fix a clique tree T corresponding to G and we root it at a vertex R (for example, let R be the clique that has vertex 1 in its residual set), obtaining T_R. For a vertex u in $V(G)$, let C_u be the clique of T_R that contains u in its residual set. We define the *depth* of u in T_R as $d(u) := d(C_u)$, where $d(C_u)$ is the depth of C_u in T_R (that is, $d(C_u)$ is the distance of C_u from the root R).

For a pair $I, F \in \Omega_G$, we define the canonical path from I to F as follows. By Lemma 2, each connected component of $G[I \oplus F]$, the subgraph of G induced by $I \oplus F$, is a tree. Since the connected components of $G[I \oplus F]$ form a partition of $I \oplus F$, we refer to the vertex sets of the connected components as components of $I \oplus F$. We process components in $I \oplus F$ in the ascending order of their smallest vertex. We first define a start vertex for each component: For a current component D, its *start vertex* $u_D \in I \oplus F$ is the vertex with the smallest depth. If there are multiple such vertices, we pick the smallest one.

We define the canonical way to convert the current component D from I to F as follows: We process

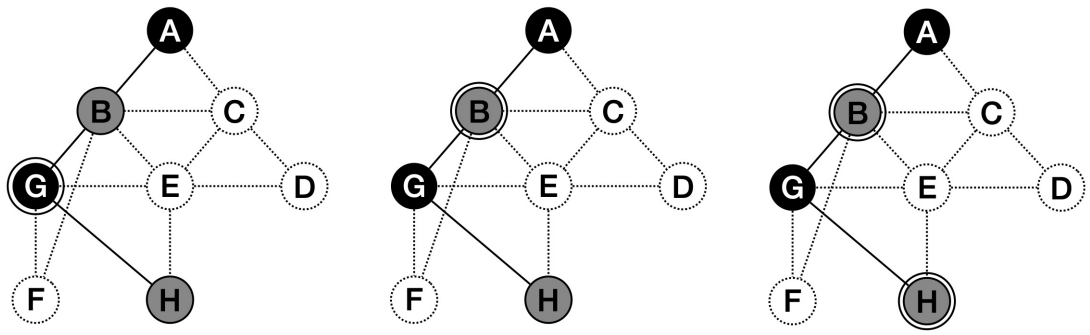

Figure 3.2: A canonical path from independent set I to F (I and F are mentioned in Figure 3.1). Notice that the doubled circles denote the current independent set. At the beginning, the current independent set consist of A, G. We first apply $\downarrow$ on A, which gives us the left independent set. Then we apply $\leftrightarrow$ to G and its parent B, which leads to the middle independent set. Finally, we apply $\uparrow$ to H to get the final independent set.

D by doing depth-first search of $G[D]$ from its start vertex u_D, processing the child vertices of the current vertex in increasing order of the sizes of the subtrees associated with the child vertices. We break ties by processing smaller children first. Let u be the current vertex in the depth-first search. Then:

- If $u \in I$: If its parent has no other neighbors in the current independent set, we apply the drag transition $\leftrightarrow$ on u and its parent. Otherwise, we apply the delete transition $\downarrow$ on u.
- If $u \in F$: If u has no children, we apply the insert transition $\uparrow$ on u. Otherwise, we proceed to process the children of u.

In other words, we always remove an I-vertex before visiting its children, and we add an F-vertex (either by the insertion $\uparrow$ or by dragging $\leftrightarrow$) to the independent set after we process all its descendants. Clearly, the transitions for $u \in I$ maintain the current state as an independent set. Notice that an F-vertex is added after its I-children have been removed, and we have removed its I-parent prior to visiting this vertex; therefore, these transitions are also legal and maintain the current state as an independent set throughout the process. Figure 3.2 shows an example of the canonical paths.

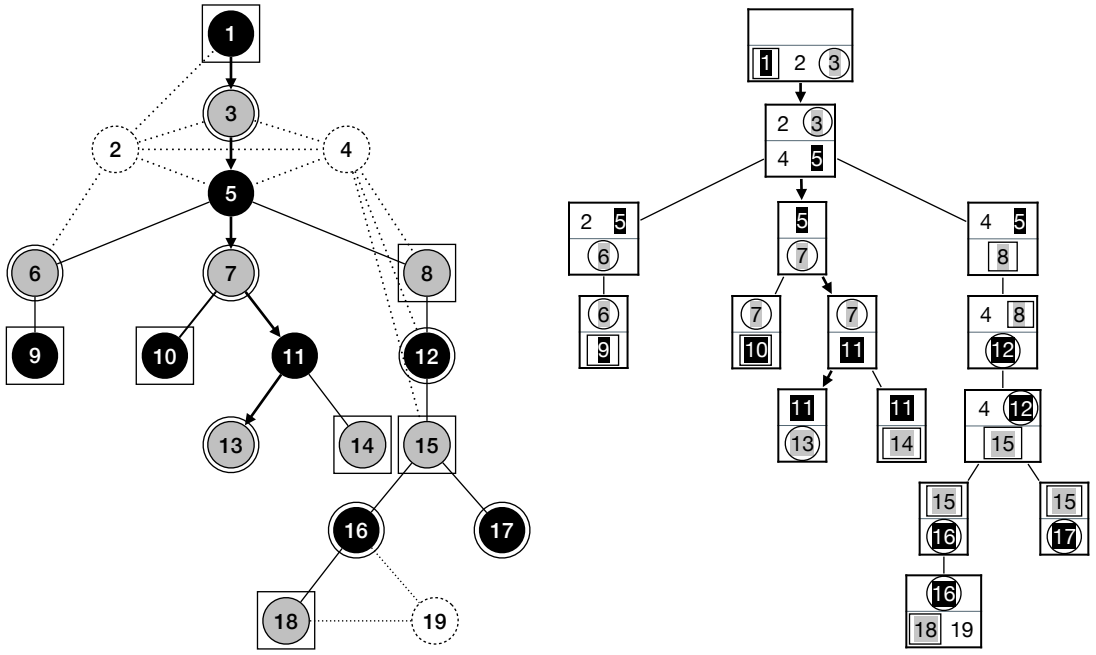

Figure 3.3: On the canonical path from I to F: On the left is a chordal graph with 19 vertices. An initial and a final independent set I and F are shown in black and grey, respectively. Solid lines indicate the edges of the induced subgraph $G[I \oplus F]$, the other edges are dotted. $G[I \oplus F]$ is processed from vertex 1, the current transition t is adding $u = 13$, and the current independent set is shown in double circles. Path p_u is shown using arrows, $Q_{p_u} = \{5, 11\}$. Vertices in $\hat{\eta}_t(I, F)$ are squared. A corresponding clique tree is on the right, each clique represented by a rectangle with the separator set and the residual set at the top and bottom, respectively.

3.2.2 Bounding the congestion

Let $t = (S, S')$ be a transition for which we want to bound the congestion $\rho(\Gamma, t)$, see (2.4), the definition of which involves a sum through all canonical paths that use t. To bound this sum, one typically defines an encoding for each canonical path $\gamma_{I,F}$ through t. The goal for the encoding is to comprise of a state of Ω_G (the set of all independent sets on G) and possibly some additional information chosen from a set of polynomial size.

Suppose $I, F \in \Omega_G$ are such that $\gamma_{I,F}$ uses t. Our encoding $\eta_{I,F}$ of $\gamma_{I,F}$ will consist of multiple parts. We start by defining its first part $\hat{\eta}_t(I, F)$, see Figure 3.3:

- Let D be the component of $I \oplus F$ on which t is applied (that is, t inserts, deletes, or drags a vertex $u \in D$).
- Let $p_u = (u_D, \ldots, u)$ be the path from the start vertex u_D to u in $G[D]$.
- Let Q_{p_u} be the set of vertices in $p_u \setminus \{u\}$ that (a) have more than one child in the tree $G[D]$ rooted at u_D, and (b) their successor vertex on p_u is not their last child.
- Then, let

$$\hat{\eta}_t(I, F) = (I \oplus F \oplus (S \cup S')) \setminus Q_{p_u}. \tag{3.1}$$

Denote by

$$\mathrm{cp}(t) := \{(I, F) \mid t \in \gamma_{I,F}\}$$

the set of pairs $(I, F) \in \Omega_G^2$ whose canonical path $\gamma_{I,F}$ uses transition t. Then we have the following lemma.

Lemma 3. *For a transition t and an independent set pair (I, F) such that $(I, F) \in \mathrm{cp}(t)$, $\hat{\eta}_t(I, F)$ is an independent set.*

Proof. Let S, S', u, and D be defined as above. Let $A = I \oplus F \oplus (S \cup S')$. Then by the definition of the canonical paths, components of $I \oplus F$ prior to D have been already processed, that is, for every such component D', the current state S (and S') contains vertices in $D' \cap F$. Likewise, every component D' after D is untouched, that is S (and S') contains vertices in $D' \cap I$. Thus, A, and therefore also $\hat{\eta}_t(I, F)$, contains the I-vertices in the processed components and the F-vertices in the untouched components. These I-vertices in A (and $\hat{\eta}_t(I, F)$) form an independent set since $I \in \Omega_G$, the same is true for the F-vertices. Moreover, if D' and D'' are two different components of $I \oplus F$ (that correspond to two different connected components of $G[I \oplus F]$), there is no edge in $G[I \oplus F]$ connecting D' and D''. Thus, so far, $A \setminus D$ forms an independent set, whose vertices do not neighbor D.

It remains to analyze D itself. The path p_u splits the tree $G[D]$ into processed parts and untouched parts. Then A agrees with I on the processed parts and it agrees with F on the untouched parts. If $v, v' \in D$ are adjacent and neither is on p_u, then v and v' cannot be both in A because these two vertices are either both in the processed or both in the untouched part of the tree. Therefore, for any two adjacent vertices in D that are both in A, at least one of them is on p_u. We will prove that it is sufficient to remove Q_{p_u} from A to make it an independent set.

Let v be a vertex in A and $v \in p_u \setminus Q_{p_u}$ (that is, v's last child in D is on p_u). Then, v must be in I because if v were in F, it would have been added to S (and therefore not be in A) by the drag transition $\leftrightarrow$ when processing v's last child. We will show that there is no neighbor v' of v in $A \setminus Q_{p_u}$, which will conclude our proof of $\hat{\eta}_t(I, F) = A \setminus Q_{p_u}$ being an independent set. Since $v \in I$, every neighbor of v in D is in F. We first consider v's neighbors on p_u: Let v_{parent} be the parent of v (if available) and v_{child} be the last child of v. Notice that $u \neq v$ because $u \in S \cup S'$ and, therefore, it is not in A. Thus, v_{child} exists.

We claim that a vertex v' is in $A \cap F \cap p_u$ if and only if it is in $Q_{p_u} \cap F$. This is because $v' \in A \cap F \cap p_u$ if and only if $v' \in F \cap p_u$ has not yet been added to S, which means that when we were processing the child v'' of v' on p_u, the transition over v'' was remove $\downarrow$ and thus v'' was not the last child of v', which is equivalent to $v' \in Q_{p_u} \cap F$. Therefore, for neighbor $v' \in \{v_{\text{parent}}, v_{\text{child}}\}$ of v we have $v' \in F$ and $v' \notin A \setminus Q_{p_u}$.

Finally, consider a neighbor v' of v in $A \setminus p_u$. Since $v_{\text{child}}\}$ is the last child of v, we have that v' has been already processed. Since $v' \in F$, it has been already added to S. Therefore, $v' \notin A$, concluding the proof that $\hat{\eta}_t(I, F) = A \setminus Q_{p_u}$ is an independent set. □

Next we observe the following bound on the size of Q_{p_u}. A similar argument was made by Ge and Štefankovič [24].

Lemma 4. *Let $t = (S, S')$ be a transition, $(I, F) \in \text{cp}(t)$, and u, D, u_D, and Q_{p_u} be defined as above. Then, $|Q_{p_u}| \leq \log_2 n$.*

Proof. Let $Q_{p_u} = \{v_1, v_2, \ldots, v_\ell\}$, where the vertices are ordered by increasing distance from u_D on the path p_u. For a vertex $v \in D$, let T_v be the v-rooted subtree of $G[D]$ and $|T_v|$ be the number of vertices in T_v. Because for each vertex v_i in Q_{p_u} its successor vertex v' on p_u is not its last child, and we always process the children in increasing order of the sizes of their subtrees, we have that $|T_{v_i}| \geq 2|T_{v'}| \geq 2|T_{v_{i+1}}|$. Therefore, $n \geq |T_{v_1}| \geq 2^\ell$, which implies $\ell \leq \log_2 n$. □ □

In our congestion bounds, we will view $I \oplus F$ through the lens of the clique tree T. The following observation follows directly from I and F being independent sets.

Observation 1. *Every clique in T can contain at most two vertices of $I \cup F$. If it contains a vertex in $I \cap F$, then it does not contain any other vertex of $I \cup F$. If it contains two vertices in $I \cup F$, then one must be in I and one in F.*

Next, we relate components of $I \oplus F$ to subtrees of the rooted clique tree T_R. Recall that C_u refers to the clique in $V(T_R)$ that contains $u \in V(G)$ in its residual set, that is $u \in \text{Res}(C_u)$.

Lemma 5. *Let D be a component of $I \oplus F$ and let X be the subtree of T spanned by clique set $\{C_u \mid u \in D\}$. Let X^R be the corresponding rooted tree of X, with edge directions consistent with T_R. Then the following holds:*

(i) C_{u_D} is the root of X^R.

(ii) Let $v \in D$ and let $p = (u_D = u_0, \ldots, u_\ell = v)$ be the path from u_D to v in $G[D]$. Then the directed path p_C in X^R from C_{u_D} to C_v passes through $C_{u_0}, C_{u_1}, \ldots, C_{u_\ell}$ in this order. Moreover, C_{u_i}'s are all distinct, with a possible exception of $C_{u_0} = C_{u_1}$.

Proof. We begin by proving (i). By contradiction, assume that a clique vertex $R' \neq C_{u_D}$ is the root of X^R. Then there is a directed path p' from R' to C_{u_D} in X^R. Since $u_D \in D$ is the vertex of the smallest depth, none of the cliques on the path p' contain a vertex in D in their residual set. Therefore, the only reason why R' would be included in X^R is that there is another vertex $w \in D$ such that the path from C_{u_D} to C_w in X goes through R'. Let p'' be the path from R' to C_w in X^R. Notice that p'' intersects p' only at R'. If u_D were of depth 0, we would have $C_{u_D} = R = R'$. Therefore, the depth of u_D, and thus also of w, is at least 1. Since both $u_D, w \in D$, there is a path from u_D to w in $G[D]$. Then, this path needs to pass through the separator set $\text{Sep}(C_{u_D})$, which means that $\text{Sep}(C_{u_D})$ contains a vertex $u' \in D$. But then u' is in the parent clique of C_{u_D}, which would mean that u' has a smaller depth than u_D. This is a contradiction and, therefore, R' must be equal to C_{u_D}.

To prove (ii), we will use induction on the depth of v (which we defined as the depth of C_v in T_R). For the base case, when v is of the same depth as u_D, we have two possibilities. If $v = u_D$, then $p = (u_D)$ and $p_C = (C_{u_D})$ and the statement holds. If $v \neq u_D$, since v and u_D are of the same depth and, by (i), C_{u_D} is the root of X^R, it follows that $C_v = C_{u_D}$. Then $p = (u_D, v)$ and $p_C = (C_{u_D})$ and the statement holds.

For the inductive claim, let v be of depth larger than u_D. If $\text{Sep}(C_v)$ contains u_D, then $p = (u_D, v)$ and p_C starts at C_{u_D} and ends at C_v, so the statement holds. Otherwise, $\text{Sep}(C_v)$ separates u_D from v. Therefore, there must be u_k, where $k \in \{1, \ldots, \ell - 1\}$, such that $u_k \in \text{Sep}(C_v)$. We will show that $k = \ell - 1$, that is, $u_{\ell-1} \in \text{Sep}(C_v)$. By contradiction, suppose that $k < \ell - 1$. By the observation 1, there are at most two vertices of D in C_v. Therefore, C_v contains u_k and v, and not $v_{\ell-1}$. But since u_k and v are in the same clique C_v, there is an edge between them. Therefore,

$u_k, u_{k+1}, \ldots, u_{\ell-1}, u_\ell = v$ is a cycle, contradicting Lemma 2 which states that $G[D]$ is a tree. Thus, $k = \ell - 1$.

The subtree of X of cliques containing $u_{\ell-1}$ has its root $C_{u_{\ell-1}}$ at smaller depth than C_v, since this subtree contains C_v. Therefore, the path p_C needs to pass through $C_{u_{\ell-1}}$. Since the depth of $u_{\ell-1}$ is smaller than the depth of v, we may use the inductive hypothesis for $v' := u_{\ell-1}$. We get that the directed path p'_C in X^R from C_{u_D} to $C_{v'}$ passes through $C_{u_0}, C_{u_1}, \ldots, C_{u_{\ell-1}}$ in this order. Since p_C passes through $C_{v'}$, it is formed by extending p'_C to C_v. Therefore, p_C passes through $C_{u_0}, C_{u_1}, \ldots, C_{u_\ell}$ in this order. Moreover, $C_{u_{\ell-1}} \neq C_{u_\ell}$, finishing the proof. □

Corollary 5.1. *Let D be a component of $I \oplus F$, let $v \in D$, and let $p = (u_D = u_0, \ldots, u_\ell = v)$ be the path from u_D to v in $G[D]$. Then the following holds for the directed path $p_C = (C_{u_D} = C_0, \ldots, C_{\ell'} = C_v)$ in T_R from C_{u_D} to C_v:*

(i) For every $i \in \{0, \ldots, \ell\}$, there exist j_i, k_i, $0 \leq j_i \leq k_i \leq \ell'$ such that the cliques on the path p_C that contain u_i are exactly cliques $C_{j_i}, C_{j_i+1}, \ldots, C_{k_i}$. Moreover, $u_i \in \mathrm{Res}(C_{j_i})$.

(ii) For every $i \in \{1, \ldots, \ell\}$, $u_{i-1} \in \mathrm{Sep}(C_{j_i})$.

Proof. We first prove (i). From the lemma we know that p_C passes through cliques $C_{u_0}, C_{u_1}, \ldots, C_{u_\ell}$ in this order. Therefore, for every $i \in \{0, \ldots, \ell\}$, there is j_i such that $C_{u_i} = C_{j_i}$. Also, by the definition of C_{u_i}, we have that $u_i \in \mathrm{Res}(C_{u_i}) = \mathrm{Res}(C_{j_i})$, which also implies that clique C_{j_i-1} (if it exists) does not contain u_i. Let $k_i \geq j_i$ be the largest index such that each of the cliques $C_{j_i}, C_{j_i+1}, \ldots, C_{k_i}$ contains u_i. It remains to show that there is no $j' \in \{0, \ldots, j_i-2\} \cup \{k_i+2, \ldots, \ell'\}$ such that $u_i \in C_{j'}$. By contradiction, suppose such j' exists. Then there is a unique path in T from $C_{j'}$ to C_{j_i}—this path is a subpath of p_C. By the induced subtree property, every clique on this path needs to contain u_i, contradicting the fact that C_{j_1-1} and C_{k_i+1} do not contain u_i. Thus, (i) holds.

For part (ii), since by the lemma $C_{u_{i-1}}$ precedes C_{u_i} on p_C, we have that $j_{i-1} < j_i$. Therefore, it suffices to show that $k_{i-1} \geq j_i$, which then, by definition, implies $u_{i-1} \in \mathrm{Sep}(C_{j_i})$. Suppose, by contradiction, that $k_{i-1} < j_i$. Since there is an edge (u_{i-1}, u_i), there must exist a clique C' containing both u_{i-1} and u_i. Since $u_i \in C'$, by Property 2.1.2 in Section 2.1.2, clique C' must be in the rooted subtree $T_{C_{u_i}}$. But then the unique path from $C_{k_{i-1}}$ to C' in T_R passes through C_{u_i}, which, by the induced subtree property, implies that $C_{k_{i-1}+1}$ contains u_{i-1} (since C' contains it), contradicting the definition of k_{i-1} as the largest index with the property described in part (i). Therefore, $k_{i-1} \geq j_i$ and (ii) also holds. □

We are ready to define the encoding of the canonical path from I to F, passing through a transition $t = (S, S')$. Let $\mathbb{N}_b := \{1, \ldots, b\}$, where b is the separator bound of G. The encoding $\eta_t(I, F)$ consists of an independent set, a vertex, and a vector from $\mathbb{N}_b^{\lfloor \log n \rfloor}$:

$$\eta_t(I, F) := (\hat{\eta}_t(I, F), u_D, s_1, s_2, \ldots, s_{\lfloor \log n \rfloor}),$$

where the vectors's role is to indicate the vertices of Q_{p_u} that were removed from $I \oplus F \oplus (S \cup S')$ during the construction of $\hat{\eta}_t(I, F)$, see (3.1). We define each s_x, $x \in \{1, \ldots, \lfloor \log n \rfloor\}$, as follows. We apply Corollary 5.1 to the path $p = p_u = (u_D = u_0, \ldots, u_\ell = u)$. For each vertex $u_i \in Q_{p_u}$, we have that $u_i \in \mathrm{Sep}(C_{j_{i+1}})$ (notice that $u \notin Q_{p_u}$, thus j_{i+1} is always well-defined). Suppose we ordered Q_{p_u} in increasing order of distance from u. Let x be the position of u_i in this ordering, thus s_x will encode u_i. Since $|\mathrm{Sep}(C_{j_{i+1}})| \leq b$, we can specify u_i by its position in $\mathrm{Sep}(C_{j_{i+1}})$. Thus, s_x is such that u_i is the s_x-th smallest vertex in $\mathrm{Sep}(C_{j_{i+1}})$. Notice that we will need as many s_x's as is the size of Q_{p_u}, which is bounded by $\lfloor \log n \rfloor$ by Lemma 4. For $x > |Q_{p_u}|$, we let $s_x = 1$. Notice that $V \times \mathbb{N}_b^{\lfloor \log n \rfloor}$ is the set B from the description of canonical paths in the preliminaries.

Lemma 6. *Let t be a transition of the Markov chain. The above-described function $\eta_t : \mathrm{cp}(t) \to \Omega_G \times V \times \mathbb{N}_b^{\lfloor \log n \rfloor}$ is injective.*

Proof. To prove the injectivity, we need to show that given a state $\hat{\eta}_t(I, F)$, a vertex u_D, a vector $(s_1, s_2, \ldots, s_{\lfloor \log n \rfloor})$, and the current transition $t = (S, S')$, we can uniquely recover the initial and final independent sets I and F.

Suppose we know $I \oplus F$. We show that then we can uniquely recover I and F. By Lemma 2, each of the components of $I \oplus F$ forms a tree. The canonical path processes the components in a specific canonical order that we can recover from $I \oplus F$. Let u be the vertex involved in t and let D be the component of $I \oplus F$ containing u. At the time of transition t, components prior to D have F-vertices in S (and S') and components after D have I-vertices in S (and S'). Similarly, $\hat{\eta}_t(I, F)$ includes the I-vertices from the prior components and the F-vertices from the latter components. Therefore, it remains to determine which vertices of D are in I and which are in F. Based on the transition t we determine whether $u \in I$ or $u \in F$ (u is in I if it is being removed and u is in F if it is being added in t). Since $G[D]$ is a tree, we know that vertices at even distance from u in $G[D]$ are in the same set as u, and the other vertices are in the other set, determining the partition of D into I- and F-vertices. Finally, $I \cap F$ can be derived from $S \backslash (I \oplus F)$.

It remains to recover $I \oplus F$. Notice that $I \oplus F = (\hat{\eta}_t(I, F) \oplus (S \cup S')) \cup Q_{p_u}$. Therefore, the only missing part in order to determine $I \oplus F$ is Q_{p_u}. Let $B = \hat{\eta}_t(I, F) \oplus (S \cup S')$. Since u_D is given and u is known from t, we can construct the path p_C from Corollary 5.1 applied to $p := p_u$. Notice

that we do not yet have the path p_u constructed—if we did, we would get $I \oplus F$ as $B \cup p_u$ and we would not need to reconstruct Q_{p_u}—but, despite not having p_u, we can construct p_C uniquely just from u and u_D. Our next step will be to construct p_u.

Let $p_C = (C_{u_D} = C_0, \ldots, C_{\ell'} = C_u)$. We want to reconstruct $p_u = (u_D = u_0, u_1, \ldots, u_\ell = u)$. We know all the vertices in B and we have that $u \in C_u$. We will work our way backwards, reconstructing u_i for $i = \ell - 1, \ell - 2, \ldots, 1$. Suppose we know u_{i+1} and so far $x - 1$ vertices of Q_{p_u} have been reconstructed. We consider $C_{u_{i+1}}$. By Corollary 5.1(ii), we have that $u_i \in \mathrm{Sep}(C_{u_{i+1}})$. By observation 1 we know that u_i and u_{i+1} are the only two vertices of $I \oplus F$ in $C_{u_{i+1}}$. Therefore we start by checking if clique $C_{u_{i+1}}$ contains a vertex from B in its separator set. If yes, it must be u_i. If not, we will use s_x to define u_i as the s_x-th smallest vertex in $\mathrm{Sep}(C_{u_{i+1}})$. This process uniquely determines p_u, and hence also $I \oplus F$, from which we obtain I and F, completing the proof. □

Lemma 7. *For every transition $t = (S, S')$ and every pair $(I, F) \in \mathrm{cp}(t)$,*

$$\pi(I)\pi(F) \leq n\bar{\lambda}^{|Q_{p_u}|+1}\pi(S)P(S, S')\pi(\hat{\eta}_t(I, F))$$

where $\bar{\lambda} := \max\{1, \lambda\}$, and Q_{p_u} and $\hat{\eta}_t(I, F)$ are defined at the start of Section 3.2.2.

Proof. Consider the expressions

$$\lambda^{|I|}\lambda^{|F|} \quad \text{and} \quad \lambda^{|S\cup S'|}\lambda^{|\hat{\eta}_t(I,F)|}.$$

Each vertex $v \in V$ contributes a factor of 1, λ or λ^2 to $\lambda^{|I|}\lambda^{|F|}$, according to whether v is in neither, exactly one, or both of I and F. If $v \notin I$ and $v \notin F$, then $v \notin S \cup S'$ and $v \notin \hat{\eta}_t(I, F)$, so the contribution to both expressions is 1. If $v \in I$ and $v \in F$, then $v \in S \cup S'$ and $v \in \hat{\eta}_t(I, F)$, so the contribution to both expressions is λ^2.

If $v \in I \oplus F$, then we have two possibilities: v can be either in $(S \cup S') \oplus \hat{\eta}_t(I, F)$, or not. In the first case v contributes λ to both expressions. In the second case, since $I \oplus F = ((S \cup S') \oplus \hat{\eta}_t(I, F)) \cup Q_{p_u}$, we get

$$\lambda^{|I|}\lambda^{|F|} \leq \lambda^{|S\cup S'|}\lambda^{|\hat{\eta}_t(I,F)|}\bar{\lambda}^{|Q_{p_u}|}.$$

Dividing by $Z_G(\lambda)^2$, the square of the partition function, and combining with the fact that $|S|, |S'| \geq |S \cup S'| - 1$, we have

$$\begin{aligned}\pi(I)\pi(F) &\leq \bar{\lambda}\bar{\lambda}^{|Q_{p_u}|}\min\{\pi(S), \pi(S')\}\pi(\hat{\eta}_t(I, F)) \\ &= n\bar{\lambda}\bar{\lambda}^{|Q_{p_u}|}\pi(S)P(S, S')\pi(\hat{\eta}_t(I, F)),\end{aligned}$$

where the last equality comes from $\pi(S)P(S,S') = \pi(S')P(S',S)$ (the so-called detailed balance condition) and from observing that if $\pi(S) \leq \pi(S')$ then $P(S,S') = 1/n$. □ □

Theorem 8. *The congestion of the above-defined canonical paths is bounded by $n^{(3+\log_2 b\bar{\lambda})}\bar{\lambda}$, where b is the separator bound of G and $\bar{\lambda} := \max\{1,\lambda\}$.*

Proof.

$$\begin{aligned}
\varrho(\Gamma) &= \max_{t=(S,S')} \left\{ \frac{1}{\pi(S)P(S,S')} \sum_{I,F\in \mathrm{cp}(t)} \pi(I)\pi(F)\,|\gamma_{IF}| \right\} \\
&\leq n\bar{\lambda} \sum_{I,F\in \mathrm{cp}(t)} \pi(\hat{\eta}_t(I,F))\bar{\lambda}^{|Q_{p_u}|}\,|\gamma_{IF}| && \text{by Lemma 7} \\
&\leq n^2\bar{\lambda} \sum_{I,F\in \mathrm{cp}(t)} \pi(\hat{\eta}_t(I,F))\bar{\lambda}^{|Q_{p_u}|} && \text{since } |\gamma_{IF}| \leq n \\
&\leq n^2\bar{\lambda}^{\log_2 n+1} \sum_{I,F\in \mathrm{cp}(t)} \pi(\hat{\eta}_t(I,F)) && \text{by Lemma 4} \\
&\leq n^2\bar{\lambda}^{\log_2 n+1} \sum_{\omega\in\Omega_G} \pi(\omega) n b^{\log_2 n} && \text{by Lemma 6} \\
&\leq n^3\bar{\lambda}(b\bar{\lambda})^{\log_2 n} && \text{since } \pi \text{ is a distribution} \\
&\leq n^{(3+\log_2 b\bar{\lambda})}\bar{\lambda}.
\end{aligned}$$

□

This allows us to bound the mixing time:

Theorem 9. *Let G be a connected chordal graph with separator bound $b \in \mathbb{N}^+$, and let $\lambda \in \mathbb{R}^+$. If $\lambda < 1$, let $x = \emptyset$, otherwise, let x be a maximum independent set. The mixing time of the Dyer&Greenhill Markov chain from the start state x is $O(n^{(4+\log_2 b\bar{\lambda})})$.*

Proof. Notice that since the number of independent sets is bounded by 2^n and x is one of the most likely states (has the largest stationary probability), we get that $\pi(x) \geq 1/2^n$. Therefore, $1/\pi(x) \leq 2^n$ and the mixing time can be bounded as follows:

$$\tau_x(\epsilon) \leq \varrho(\Gamma)\left(\ln\frac{1}{\pi(x)} + \ln\frac{1}{\epsilon}\right) \leq n^{(3+\log_2 b\bar{\lambda})}\bar{\lambda}\left(n\ln 2 + \ln\frac{1}{\epsilon}\right).$$

□

We remark that obtaining the start state x, a maximum independent set, is computable in polynomial time for chordal graphs. We conclude with a natural open problem, in addition to extending rapid mixing results to other graph classes: extending our results to arbitrary chordal graphs.

Chapter 4

Sampling and Counting Graph Orientations

In Chapter 2, we mentioned that counting of acyclic orientations, bipolar orientations, and sink-free orientations are all #P-complete for general graphs, which motivates our study on chordal graphs, a restricted and well-studied graphs class. In this chapter, we present polynomial-time counting and sampling algorithms for acyclic orientations and bipolar orientations. We then provide polynomial-time counters for sink-free orientations and source-sink-free orientations. The sampling and counting of the four types of orientations are all based on dynamic programming over the clique tree structure. Finally, we introduce the work of sampling partial acyclic orientations on chordal graphs by the so-called partial rejection sampling technique.

4.1 Notation and Useful Properties

Recall that for a chordal graph G, its clique tree T_G, and one of its maximum cliques C_r, we use T_{G,C_r} to represent the clique tree that has C_r as its root. We simplify the notation to be T_{C_r} if G is clear from the context. Then for a clique C, we let $T_{C_r,C}$ denote the subtree of T_{C_r} containing C and its descendants. And we write it as T_C if C_r is clear from the context. We use $G[T_C]$ for the subgraph induced by the vertices that belong to cliques in T_C , i.e., $G[T_C] := G\left[\bigcup_{C' \in V(T_C)} C'\right]$. We will often work with the following subgraph of $G[T_C]$: Let $\hat{G}[T_C]$ be $G[T_C]$ with the edges within the separator set $\mathrm{Sep}(C)$ removed, i.e., $\hat{G}[T_C] := G[T_C] - E(G[\mathrm{Sep}(C)])$. Figure 4.1 shows an example of $G[T_C]$ and $\hat{G}[T_C]$. The following lemma will be essential for our calculations.

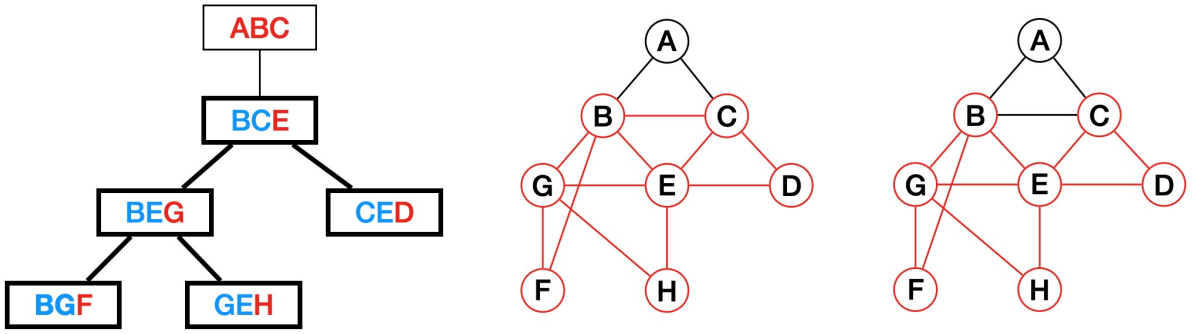

Figure 4.1: Illustrating T_C, $G[T_C]$, and $\hat{G}[T_C]$: Graph G is shown in the right two figures, and one of its clique trees, T_G is shown on the left. The bold part of the clique tree T_{ABC} is the sub clique tree T_{BCE}; the red subgraph in the middle is $G[T_{BCE}]$; the red subgraph on the right is $\hat{G}[T_{BCE}]$ because $E(G[\text{Sep}(BCE)]$ contains the edge BC.

Lemma 10. *Let C be a clique in the rooted clique tree T_{C_r} and let $C_1, C_2, \ldots, C_d$ be its children cliques. The edge sets of the graphs $\hat{G}[T_{C_i}]$, $i = 1, \ldots, d$, are mutually disjoint.*

Proof. By contradiction, suppose that there are $i \neq j \in \{1, \ldots, d\}$ such that $\hat{G}[T_{C_i}]$ and $\hat{G}[T_{C_j}]$ share an edge $e = (u, v)$. Since $\text{Sep}(C_i)$ is a separator in G, separating vertices in $V(G[T_{C_i}]) - \text{Sep}(C_i)$ from $V(G) - V(G[T_{C_i}])$, and since $V(G[T_{C_j}]) \subseteq V(G) - V(G[T_{C_i}])$, it follows that u and v must be in $\text{Sep}(C_i)$. But then e is not in $\hat{G}[T_{C_i}]$, a contradiction. □

In order to make the running times of our algorithms more readable, we assume that each arithmetic operation takes a constant time. This is, of course, a bit optimistic, since the ultimate number of orientations can be as high as 2^m for a graph with m edges, and, therefore, the true running time of each arithmetic operation adds a factor of about $m\,\text{polylog}(m)$. We use $\tilde{O}()$ notation to indicate that this factor is omitted from our running time estimate.

Our sampling algorithms produce orientations uniformly at random: Each orientation is chosen with equal probability from the set of all desired orientations. We use $[d]$ to denote $\{1, 2, \ldots, d\}$.

4.2 Sampling and Counting Acyclic Orientations

In this section, we present an exact counter of acyclic orientations, which also leads to an efficient uniform sampler. The sampler runs in linear time per sample which is, in fact, faster running time

than the counting algorithm itself.

4.2.1 Related work

The problems of counting acyclic orientations and counting and sampling sink-free orientations on a given graph have attracted a lot of attention. Linial [46] showed that counting acyclic orientations is a #P-complete problem on general graphs. The problem of counting acyclic orientations corresponds to a specific input of the well-studied Tutte polynomial $T_G(x, y)$ [14], in particular to $T_G(2, 0)$ [82]; see also Section 2.2.2 for additional discussion of the Tutte polynomial. The number of acyclic orientations on a graph G is also related to the chromatic polynomial of G evaluated at -1 colors [67]: in particular, it is $P_G(x) \cdot (-1)^{|V(G)|}$, where $P_G(x)$ is the chromatic polynomial of G. Since the chromatic polynomial of a chordal graph can be efficiently computed, see e.g., Agnarsson [1], this yields an efficient counter for acyclic orientations. However, due to the inclusion-exclusion effect of evaluating the polynomial at $x = -1$, it appears that this counting algorithm does not yield a corresponding uniform sampler. Moreover, to the best of our knowledge, the problem is not known to be self-reducible, a property that can be used to derive (almost) uniform samplers from counting algorithms (and vice versa, approximate counting algorithms from sampling algorithms).

4.2.2 Our contributions

We first describe an alternative algorithm for counting acyclic orientations of a given chordal graph, one that does not rely on the chromatic polynomial and avoids an inclusion-exclusion type of calculation. This allows us to efficiently sample acyclic orientations uniformly at random. In particular, we prove the following two theorems:

Theorem 11. *Let G be a connected chordal graph. The number of its acyclic orientations can be calculated in $\tilde{O}(|V(G)|) + O(|E(G)|)$ time.*

Theorem 12. *A uniformly random acyclic orientation can be produced in time $O(|E(G)|)$.*

One interesting feature of our sampling algorithm is that it is a stand-alone algorithm (which does not need to determine the corresponding counts), and hence its running time is more efficient than that of the counting algorithm. The theorem relies on the following lemma which explores the structure of acyclic orientations and their relation to a clique tree of the given graph.

Lemma 13. *Let G be a connected chordal graph and let T be a rooted clique tree of G. For a clique C in T and an acyclic orientation σ over C, let $\mathrm{AO}(T_C, \sigma)$ be the set of acyclic orientations on $G[T_C]$ that are consistent with σ. For any C and any two acyclic orientations σ_1 and σ_2 over C, we have*

$$|\mathrm{AO}(T_C, \sigma_1)| = |\mathrm{AO}(T_C, \sigma_2)|.$$

Proof. We will construct a bijective mapping between $\mathrm{AO}(T_C, \sigma_1)$ and $\mathrm{AO}(T_C, \sigma_2)$ for arbitrary σ_1 and σ_2. The construction will proceed by induction on the height of C in the rooted clique tree T. When C is a leaf, then $\mathrm{AO}(T_C, \sigma_1)$ contains a single orientation, σ_1 itself. Therefore, we map σ_1 to σ_2 to get the bijection.

When C is not a leaf, let $C_1, \ldots, C_d$ be its children in the clique tree. For each C_i, let σ_j^i be the orientation of $\mathrm{Sep}(C_i)$ consistent with σ_j for $j = 1, 2$. Let α be an orientation in $\mathrm{AO}(T_C, \sigma_1)$. By induction, we have a bijection between $\mathrm{AO}(T_{C_i}, \sigma_1^i)$ and $\mathrm{AO}(T_{C_i}, \sigma_2^i)$. Let α^i be the orientation of $G[T_{C_i}]$ consistent with α, and let β^i be its corresponding orientation (from the bijection) in $\mathrm{AO}(T_{C_i}, \sigma_2^i)$. We map α to the orientation β defined as the union of $\beta_1, \ldots, \beta_d$, and σ_2. We claim that:

- The union of $\beta_1, \ldots, \beta_d$ is an orientation, that is, there are no i_1, i_2 and an edge $e = (u, v)$ such that e is oriented in opposite ways in β_{i_1} and β_{i_2}. Suppose, by contradiction, there are such i_1, i_2 and e. Therefore, the edge e is in $G[T_{C_{i_1}}] \cap G[T_{C_{i_2}}]$, and as such $u, v \in \mathrm{Sep}(C_1) \cap \mathrm{Sep}(C_2)$. Then, since both $\sigma_2^{i_1}$ and $\sigma_2^{i_2}$ are consistent with σ, the orientation of e is identical in both β_{i_1} and β_{i_2}, a contradiction.
- β is an orientation. This follows since each β_i is consistent σ_2^i; the additional edges oriented by σ_2 do not interfere with the edges oriented by the union.
- $\beta \in \mathrm{AO}(T_C, \sigma_2)$. We need to show that β is acyclic. Suppose, by contradiction, that β contains a cycle. Let $u_1, \ldots, u_q$ be a shortest cycle in β. By the inductive hypothesis, each β_i is acyclic. Therefore, the cycle needs to pass through C. But since σ_2 is acyclic, the cycle also needs to contain vertices outside of C. If the cycle contained exactly one vertex of C, it would be entirely within some T_{C_i}, a contradiction with β_i being acyclic. Since the cycle is shortest, it contains at most two vertices of C (if it contained more, the cycle could have been shortened by skipping over one of these vertices, since C is a clique). Therefore, the cycle contains exactly two vertices of C, which have to be consecutive on C (otherwise the cycle could have been shortened). Without loss of generality, let these vertices be u_1 and u_2. Then, let i be such that u_3 is in $G[T_{C_i}]$. Notice that there is a unique such i, since $u_3 \notin C$.

Therefore, $u_2 \in \mathrm{Sep}(C_i)$. Let u_p be the last vertex on the cycle in $G[T_{C_i}] - \mathrm{Sep}(C_i)$. Then, $p = q$ since the cycle needs to pass through $\mathrm{Sep}(C_i)$ again but there is only one other vertex in C on the cycle: u_1. But then $u_1 \in \mathrm{Sep}(C_i)$ and the cycle is entirely included in $G[T_{C_i}]$, a contradiction with β_i being acyclic.

- The mapping is injective. Suppose there is a β mapped to by two different orientations in $\mathrm{AO}(T_C, \sigma_1)$, let us call them α and α'. Since β is the union of $\beta_1, \ldots, \beta_d$ and σ_2, there must be a β_i that both α_i and α'_i map to, a contradiction with the inductive hypothesis.
- The mapping is surjective. This follows since for a given $\beta \in \mathrm{AO}(T_C, \sigma_2)$, we can reverse the process and construct a corresponding $\alpha \in \mathrm{AO}(T_C, \sigma_1)$ that maps to β.

Therefore, the mapping is a bijection, completing the proof. □

Proof of Theorem 11. Let T be a clique tree of G rooted at a clique C_r. For a clique C in T, we define $\mathrm{AO}(T_C)$ as the number of acyclic orientations of $G[T_C]$ under the assumption that the orientation of the edges of $G[\mathrm{Sep}(C)]$ has been fixed. Then, $\mathrm{AO}(T_{C_r})$ computes the overall number of acyclic orientations of G, since $\mathrm{Sep}(C_r) = \emptyset$. We show how to compute $\mathrm{AO}(T_C)$ by dynamic programming over the clique tree:

$$\mathrm{AO}(T_C) = \frac{|C|!}{|\mathrm{Sep}(C)|!} \prod_{i=1}^{d} \mathrm{AO}(T_{C_i}), \tag{4.1}$$

where $C_1, \ldots, C_d$ are the children cliques of C in T (and $d = 0$ if C is a leaf of T). To prove the correctness of this expression, let $\sigma_{\mathrm{Sep}(C)}$ be the given orientation of $G[\mathrm{Sep}(C)]$. We first extend it to an acyclic orientation σ_C over C: Since an acyclic orientation of a clique corresponds to an ordering of its vertices, we have $|C|!$ such orderings but we are overcounting by allowing different orderings within $\mathrm{Sep}(C)$. Hence, the number of σ_C's consistent with $\sigma_{\mathrm{Sep}(C)}$ is $\frac{|C|!}{|\mathrm{Sep}(C)|!}$. Therefore, the calculation is correct if C is a leaf of T.

For a non-leaf C, let us fix an arbitrary σ_C and let $\sigma_{\mathrm{Sep}(C_i)}$ be the orientation restricted to $G[\mathrm{Sep}(C_i)]$. Let A_i be the set of acyclic orientations of $G[T_{C_i}]$ consistent with $\sigma_{\mathrm{Sep}(C_i)}$. We show that there is a bijection between $\mathrm{AO}(T_C, \sigma_C)$ and $A_1 \times \cdots \times A_d$. For an orientation $\sigma \in \mathrm{AO}(T_C, \sigma_C)$, let σ_i be the orientation restricted to $G[T_{C_i}]$. This mapping is clearly an injection since a subset of an acyclic orientation is still acyclic and we can reconstruct the original orientation from the σ_i's. To prove its surjectivity, let $\sigma_i \in A_i$ for $i \in [d]$. We construct σ by taking the union of the σ_i's and σ_C. It follows from Lemma 10 that every edge is oriented consistently (no edge is oriented in two opposite directions), σ is consistent with σ_C by construction. It remains to show that σ is acyclic.

By contradiction, suppose it contains a cycle; we will consider a shortest cycle in σ. Since the σ_i's and σ_C are acyclic, the cycle must pass through multiple $G[T_{C_i}]$'s, visiting C at least twice. Let $u_1, u_2 \in C$ be two vertices on this cycle. There is an edge in C between u_1 and u_2, hence the cycle can be shorten in σ by taking a shortcut through this oriented edge, a contradiction. Therefore $\sigma \in \mathrm{AO}(T_C, \sigma_C)$. By construction this σ maps to the σ_i's, proving the surjectivity, and hence the bijection, of the mapping.

Notice that $\mathrm{AO}(T_{C_i}) = |A_i|$. Since, by the inductive hypothesis, the calculation of $\mathrm{AO}(T_{C_i})$ is correct, it follows that $\prod_{i=1}^{d} \mathrm{AO}(T_{C_i})$ computes $|\mathrm{AO}(T_C, \sigma_C)|$. By Lemma 13, $|\mathrm{AO}(T_C, \sigma_C)|$ does not depend on the specific orientation of σ_C, and there are $\frac{|C|!}{|\mathrm{Sep}(C)|!}$ possible σ_C's, yielding the expression (4.1).

After constructing the clique tree T (which is of size $O(|V(G)|)$), the algorithm performs a tree traversal of T, and at each clique C of T it performs $O(\deg_T(C)+1)$ arithmetic operations.[1] Hence the running time is $\tilde{O}(|V(G)|) + O(|E(G)|)$. □

An example of calculation of Equation 4.1 is shown in Figure 4.2.

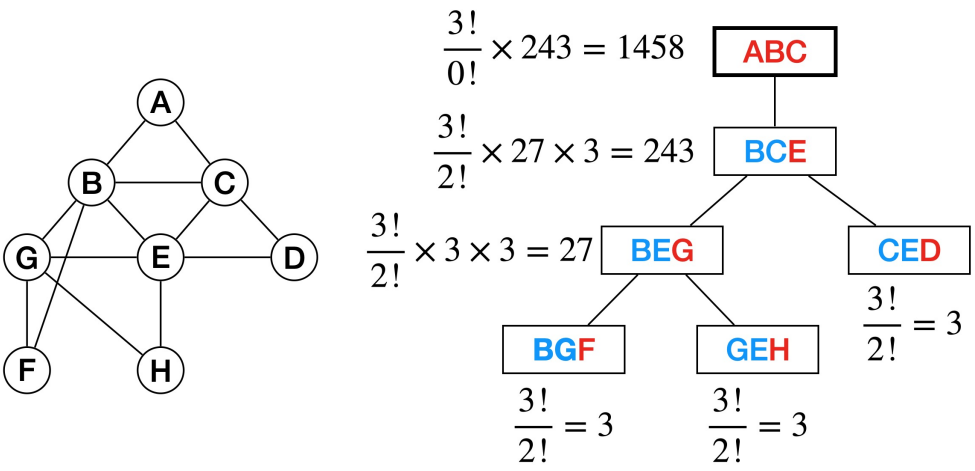

Figure 4.2: Counting of acyclic orientations in a graph G. When the current clique $\mathbb{C}$ is among $\{\{B,G,F\},\{G,E,H\},\{C,E,D\}\}$, since it does not have any children, then $\mathrm{AO}(T_{\mathbb{C}}) = \frac{|\mathbb{C}|!}{|\mathrm{Sep}(\mathbb{C})|!} = \frac{3!}{2!} = 3$. When $\mathbb{C} = \{B,E,G\}$, $\mathrm{AO}(T_{\mathbb{C}}) = \frac{3!}{2!} \cdot \mathrm{AO}\left(T_{\{B,G,F\}}\right) \cdot \mathrm{AO}\left(T_{\{G,E,H\}}\right) = 27$. Calculating each quantity by this manner will finally give us the number of acyclic orientations in G, i.e., $\mathrm{AO}\left(T_{\{A,B,C\}}\right) = 1458$.

[1] Computation of the factorial of an n-bit number takes $O(n \operatorname{polylog} n)$ [13], which will be subsumed by our $\tilde{O}()$ notation.

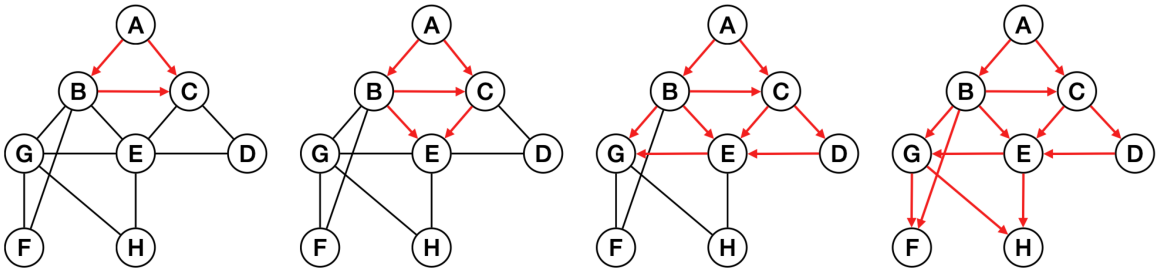

Figure 4.3: Sampling acyclic orientations on graph G. We first assign clique $\{A, B, C\}$ an orientation uniformly at random (in this example we pick $A \to B$, $A \to C$, and $B \to C$). Then we pick an acyclic orientation that is consistent with $B \to C$ over $\{B, C, E\}$ uniformly at random, which is $B \to C, B \to E, C \to E$. Keep doing it for all cliques in a depth-first search manner gives us an acyclic orientation on G.

Proof of Theorem 12. To sample an acyclic orientation uniformly at random, we first construct a clique tree of the input graph and randomly pick a clique C_r as the root. We pick a uniformly random ordering of C_r. Then we process the remaining cliques in a depth-first search manner. Let C be the current clique we are processing, we always pick an orientation on $G[C]$ that is consistent with $G[\mathrm{Sep}(C)]$ (see Figure 4.3). This can be done by choosing a random ordering π of C and replacing the relative order of $\mathrm{Sep}(C)$ in π by the given ordering. Hence the running time is $O(\sum_{C \in T} |C|)$. Observe that $\sum_{C \in T} |C| = \sum_{v \in V(G)} |S_v|$, where S_v is the set of cliques of T that contain v. Suppose the cliques in S_v are $C_1, \ldots, C_k$. Then there are vertices $v_1, \ldots, v_k$ such that $C_{v_i} = C_i$. Since $v_i \in \mathrm{Res}(C_i)$ by definition, and since each vertex is in a unique residual set, it follows that $v_1, \ldots, v_k$ are all distinct (one of which could be v). Also, they are each a neighbor of v (or v itself). Therefore, $|S_v| = k \leq \deg_G(v) + 1$. Then, $\sum_{v \in V(G)} |S_v| \leq \sum_{v \in V(G)} \deg_G + 1 = O(|E(G)|)$. □

4.3 Sampling and Counting Bipolar Orientations

In this section, we present efficient (exact) counting and sampling algorithms for bipolar orientations on chordal graphs, extending our approach for acyclic orientations to maintain the desired source and sink. Recall that a bipolar orientation of an undirected graph is an assignment of a direction to each edge that makes the graph directed, acyclic, and with a single source and a single sink.

4.3.1 Related works

Bipolar orientations are also known as *st*-numberings and are a natural restriction of acyclic orientations with applications in, for example, planarity testing [45]. While finding a bipolar orientation

can be done in linear time by depth first search [72], to the best of our knowledge no efficient algorithm for counting or sampling of bipolar orientations is known, even for restricted graph classes. The exact counting problem is #P-complete on general graphs due to a simple reduction from counting acyclic orientations: for a graph G, add two new vertices, make them the source and the sink, and connect them to each vertex in G.

4.3.2 Our result

In this section we prove the following theorem:

Theorem 14. *Let G be a chordal graph and $s \neq t$ be two of its vertices. The number of bipolar (s,t)-orientations of G can be computed in $\tilde{O}(|V(G)|) + O(|E(G)|))$ time. A uniformly random bipolar (s,t)-orientation of G can be produced in time $O(|E(G)|)$.*

We first prove a couple of lemmas that characterize bipolar orientations in chordal graphs. Recall that for a vertex $v \in V(G)$ we use C_v to denote the unique clique in T with $v \in \mathrm{Res}(C_v)$.

Lemma 15. *Let G be a connected chordal graph and let $s, t \in V(G)$, $s \neq t$. Let T be a clique tree of G rooted at a clique that contains s and let C be a maximal clique of G. Let σ be a bipolar (s,t)-orientation of G, and let σ' be the orientation σ restricted to $G[T_C]$. Then the following holds:*

(i) If C is not a descendant of C_t, and is not on the C_s-C_t path in T, then σ' is bipolar with both its source and its sink in $\mathrm{Sep}(C)$.

(ii) If C is on the C_s-C_t path in T and $C \neq C_s$, then σ' is bipolar where its source is in $\mathrm{Sep}(C)$ and its sink is t. On the other hand, if $C \neq C_t$, then σ' restricted to $G[C]$ has its sink in $C \cap \mathrm{Sep}(C')$, where C' is the child of C on the C_s-C_t path.

(iii) If C is a descendant of C_t in T and $C \neq C_t$, then σ' is bipolar with both its source and its sink in $\mathrm{Sep}(C)$.

Proof. Clearly, σ' is acyclic, so we only focus on proving bipolarity and the locations of the source and the sink. Whichever case applies, $C \neq C_s$. We first prove that there is no source $s' \notin \mathrm{Sep}(C)$ in σ'. Suppose such s' exists. Since G is connected, there is a path between s and s', and every such path goes through the separator $\mathrm{Sep}(C)$. Since s is the only source in σ, there is a directed path in σ from s to every vertex in G. Therefore, there has to be a directed path from s to s' consistent with σ, and the path passed through $\mathrm{Sep}(C)$. Thus, there is an incoming edge to s' in

$G[T_C]$, a contradiction with s' being a source in σ'. We proved that all sources in σ' are in $\mathrm{Sep}(C)$. Moreover, since $\mathrm{Sep}(C)$ is a clique, there can be at most one source (and at most one sink) in $\mathrm{Sep}(C)$.

For the number and location of the sinks, we distinguish the individual cases. We first prove (i). The assumption implies that $t \notin V(G[T_C])$. Suppose, by contradiction, that σ' contains a sink $t' \notin \mathrm{Sep}(C)$. Since σ is bipolar, for every vertex $v \in V(G)$ there has to be a directed path in G consistent with σ from v to t. Consider $v = t'$. Every path from t' to t in G has to go through $\mathrm{Sep}(C)$, which means that t' has to have an outgoing edge in σ and therefore cannot be a sink in σ', a contradiction. Therefore, in case (i) σ' is bipolar with both its source and its sink in $\mathrm{Sep}(C)$.

For part (ii), we have that $t \in V(G[T_C])$. Therefore, t must be a sink in σ'. Suppose, by contradiction, that there is another sink t' in σ'. Since $G[T_C]$ is connected, there is a path from t' to t in $G[T_C]$, and therefore also in G. Since t is the only sink in σ, this path has to be oriented from t' to t, and hence t' cannot be a sink in σ, not in σ'. Thus, t is the only sink in σ'. If $C \neq C_t$, let σ'' be the restriction of σ' to $G[C]$ and let s'' and t'' be the source and the sink of σ''. Since σ' is bipolar, there must be a directed path from t'' to t in σ'. Then, prepending this path by the edge (s'', t''), we get a path from s'' to t'' in G. This path must pass through $\mathrm{Sep}(C')$. Since t'' is a sink in σ'', the only vertices of C on this path are s'' and t''. Therefore, $t'' \in C \cap \mathrm{Sep}(C')$.

For part (iii), t may but does not have to be in $V(G[T_C])$. If $t \in V(G[T_C])$, then it must be the only sink in σ' for the same reasons as in the previous case, and we also have $t \in \mathrm{Sep}(C)$ since $C \neq C_t$. If $t \notin V(G[T_C])$, then the same argument as in the first case applies, and it follows that there is a single sink, which must be in $\mathrm{Sep}(C)$. □

Lemma 16. *Let G be a connected chordal graph and let $s, t \in V(G)$, $s \neq t$. Let T be a clique tree of G rooted at a clique that contains s and let C be a maximal clique of G that is not a leaf in T. Let $C_1, \ldots, C_d$ be the children cliques of C in T. Then the following holds:*

(i) Suppose C is not a descendant of C_t, and is not on the C_s-C_t path in T; or $C \neq C_t$ is a descendant of C_t in T. Let σ_C be an orientation of $G[C]$ and let σ_i be a bipolar orientation of $G[T_{C_i}]$ consistent with σ_C that has both its sink and its source in $\mathrm{Sep}(C_i)$, for every $i \in [d]$. Then, the orientation σ defined as taking the union of σ_C and all the σ_i's is bipolar with both its source and its sink in C (identical to the source and sink in σ_C).

(ii) Suppose C is on the C_s-C_t path in T and $C \neq C_s$. If $C = C_t$, let $t' = t$, otherwise, let C' be the next clique after C on the C_s-C_t path and let t' be any vertex in $\mathrm{Sep}(C')$. Let σ_C be an orientation of $G[C]$ with its source in $\mathrm{Sep}(C)$ and its sink at t'. Let σ_i be a bipolar orientation

of $G[T_{C_i}]$ consistent with σ_C that has both its sink and its source in $\mathrm{Sep}(C_i)$*, for every $i \in [d]$ such that $C_i \neq C'$. Finally, let σ' be a bipolar orientation $G[T_{C'}$ consistent with σ_C that has its source in* $\mathrm{Sep}(C_i)$ *and its sink at t. Then, the orientation σ defined as taking the union of σ_C, all the σ_i's, and σ' is bipolar with its source identical to that of σ_C and its sink at t.*

Proof. We first prove part (i). First, notice that σ is indeed well-defined: There is no edge oriented in opposite directions by some of the orientations that are being combined. (This is due to Lemma 10 and the σ_i's being consistent with σ_C.) We need to show that σ is acyclic. Suppose, by contradiction, that it contains a cycle. Consider a shortest cycle in σ. It must go through multiple subtrees T_{C_i} since each σ_i is acyclic. Therefore, the cycle passes through C at least twice, let us call these vertices u_1 and u_2. There is a directed edge between u_1 and u_2, allowing us to shorten the cycle, a contradiction. Since all sources and sinks are in C, the sources and sinks of σ must be in C as well and since C is a clique, the only source and sink of σ are the source and sink of σ_C.

The acyclicity argument for part (ii) is analogous, as is the location of the single source of σ. As for the sink, each σ_i has a single sink which is in C, where there is an edge from this sink to t' (unless the sink itself is t'). But since t' is in C', there is a path from t' to t in σ', showing that t is a sink of σ, and it is the only sink since σ' is bipolar. □

Proof of Theorem 14. We first fix a clique tree T of G and root it at a clique that contains s. Therefore, the root clique is C_s, the unique clique that contains s in its residual set. For each maximal clique $C \neq C_s$ of G we define the following quantities:

- $\mathrm{BPO}(T_C, w)$, where $C \neq C_s$ is on the C_s-C_t path in T, and $w \in \mathrm{Sep}(C)$: Assuming $G[\mathrm{Sep}(C)]$ has been already oriented with w as the sink of this orientation, $\mathrm{BPO}(T_C, w)$ counts the bipolar orientations of the graph $G[T_C]$ consistent with the orientation of $G[\mathrm{Sep}(C)]$ that have their source in $\mathrm{Sep}(C)$ and their sink is t.
- $\mathrm{SBPO}(T_C)$, where C is not on the C_s-C_t path in T: Assuming $G[\mathrm{Sep}(C)]$ has been already oriented, $\mathrm{SBPO}(T_C)$ counts the bipolar orientations of the graph $G[T_C]$ consistent with the orientation of $G[\mathrm{Sep}(C)]$ that have both their source and their sink in $\mathrm{Sep}(C)$.

We show how to compute these quantities by dynamic programming on the tree T, and how to use them to compute the number of bipolar (s,t)-orientations of G.

Base case. C is a leaf of T. If C is on the C_s-C_t path, then $C = C_t$. In this case,

$$\mathrm{BPO}(T_C, w) = \frac{(|C|-2)!}{(|\operatorname{Sep}(C)|-1)!}.$$

This is because the source of the orientation of $G[\operatorname{Sep}(C)]$ must remain a source (and hence is first in the ordering) and $t \notin \operatorname{Sep}(C)$ must be the sink (the last in the ordering). Thus we have $(|C|-2)!$ orderings, which are overcounting the possibilities by a factor of $(|\operatorname{Sep}(C)|-1)!$, since there are $|\operatorname{Sep}(C)|-1$ vertices we are reordering, but should have, in the ordering of the $|C|-2$ vertices.

If C is not on the C_s-C_t path, we get

$$\mathrm{SBPO}(T_C) = \frac{(|C|-2)!}{(|\operatorname{Sep}(C)|-2)!},$$

since both the source and the sink are given by the ordering of $G[\operatorname{Sep}(C)]$.

Inductive case. C is not a leaf of T; let $C_1, \ldots, C_d$ be its children cliques in T.

If C is on the C_s-C_t path, we are computing $\mathrm{BPO}(T_C, w)$. We will distinguish two cases:

1. $C \neq C_t$. Without loss of generality, assume that C_d is the clique that contains C_t in its subtree; i.e., $C_t \in V(T_{C_d})$. Then, we need to orient the remaining edges in the clique C in such a way so that, by Lemma 15, the sink is in $\operatorname{Sep}(C_d)$. We will consider two further subcases:

   a) If $w \notin \operatorname{Sep}(C_d)$, we need to choose a new sink w' in $\operatorname{Sep}(C_d)$, effectively extending the orientation of $\operatorname{Sep}(C)$ by appending w' to its end. We consider the corresponding orientations of C. We need to orient each $G[T_{C_i}]$, $i < d$, consistently with the source and the sink given by the orientation of $G[C]$. Finally, for $G[T_{C_d}]$, we need to be consistent with w' being the sink given by the orientation of $G[C]$. Therefore,

   $$\mathrm{BPO}(T_C, w) = \sum_{w' \in \operatorname{Sep}(C_d)} \frac{(|C|-2)!}{(|\operatorname{Sep}(C)|-1)!} \mathrm{BPO}(T_{C_d}, w') \prod_{i=1}^{d-1} \mathrm{SBPO}(T_{C_i}).$$

   b) If $w \in \operatorname{Sep}(C_d)$, then we have the option of choosing a new sink $w' \in \operatorname{Sep}(C_d)$ or keeping w:

   $$\mathrm{BPO}(T_C, w) = \frac{(|C|-2)!}{(|\operatorname{Sep}(C)|-2)!} \mathrm{BPO}(T_{C_d}, w) \prod_{i=1}^{d-1} \mathrm{SBPO}(T_{C_i}) + \sum_{w' \in \operatorname{Sep}(C_d), w' \neq w} \frac{(|C|-2)!}{(|\operatorname{Sep}(C)|-1)!} \mathrm{BPO}(T_{C_d}, w') \prod_{i=1}^{d-1} \mathrm{SBPO}(T_{C_i}).$$

We argue that this calculation accounts for each orientation that should have been counted in $\mathrm{BPO}(T_C, w)$ exactly once. By Lemma 15, for every orientation σ of $G[T_C]$ corresponding to $\mathrm{BPO}(T_C, w)$, we have a corresponding restriction σ_i of σ to $G[T_{C_i}]$, which has been counted in the corresponding BPO or SBPO by the inductive hypothesis. Therefore, σ is accounted for by $\mathrm{BPO}(T_C, w)$. Lemma 16, part (ii), states that each orientation arising by combining individual orientations corresponding to the expressions on the right hand side, is an orientation corresponding to $\mathrm{BPO}(T_C, w)$. This argument provides to a bijection between the respective sets and, therefore, the calculation is correct. Similar argument holds in the other cases in this proof but we will not spell it out each time for brevity reasons.

2. $C = C_t$. This is similar to the previous case, but we have to take $w' = t$ as the new sink. Notice that $t \notin \mathrm{Sep}(C)$ (by the definition of C_t) but that for each C_i, either $t \in \mathrm{Sep}(C_i)$ or $t \notin C_i$. In either case, by Lemma 15, $G[T_{C_i}]$ gets an orientation of $G[\mathrm{Sep}(C_i)]$ given and needs to orient the edges so that the source and the sink remain in $\mathrm{Sep}(C_i)$. Therefore, we get:

$$\mathrm{BPO}(T_C, w) = \frac{(|C|-2)!}{(|\,\mathrm{Sep}(C)|-1)!}\prod_{i=1}^{d}\mathrm{SBPO}(T_{C_i}).$$

If C_t is not on the C_s-C_t path in T, we need to compute $\mathrm{SBPO}(T_C)$. Then, the graphs induced by the subtrees need to be oriented consistently with the orientation of their separators, and they have to maintain the given source and the sink. Hence,

$$\mathrm{SBPO}(T_C) = \frac{(|C|-2)!}{(|\,\mathrm{Sep}(C)|-2)!}\prod_{i=1}^{d}\mathrm{SBPO}(T_{C_i}).$$

Finally, to output the number of bipolar (s,t)-orientations of G, we employ a similar logic as in the case when C is on the C_s-C_t path. Let $C_1, \ldots, C_d$ be the children cliques of C_s in T ($d = 0$ if C_s is a leaf).

1. If $C_s \neq C_t$. Then $d > 0$. Without loss of generality, assume that $C_t \in V(T_{C_d})$; then the sink needs to be in $\mathrm{Sep}(C_d)$ and s needs to be the source.

$$\mathrm{BPO}(G) = \sum_{w' \in \mathrm{Sep}(C_d)} (|C|-2)!\,\mathrm{BPO}(T_{C_d}, w')\prod_{i=1}^{d-1}\mathrm{SBPO}(T_{C_i}).$$

2. If $C_s = C_t$, we follow the argument of case 2 where the separator set is empty and s needs to be the source:

$$\mathrm{BPO}(G) = (|C|-2)!\prod_{i=1}^{d}\mathrm{SBPO}(T_{C_i}).$$

As for the running time, notice that the calculations of $\mathrm{BPO}(T_C, w)$ do not depend on w itself (this follows by induction on the tree) and therefore it suffices to store just one quantity $\mathrm{BPO}(T_C)$ for each C. Then, we are computing two quantities, $\mathrm{BPO}(T_C)$ and $\mathrm{SBPO}(T_C)$ per C. Furthermore, in the quantities that involve a summation, the summation can be replaced by a factor of $|\operatorname{Sep}(C_d)|$, or $|\operatorname{Sep}(C_d)|-1$, respectively. Therefore the running time, analogously to Theorem 11, is $\tilde{O}(|V(G)|)+O(|E(G)|)$. As for the sampling, we can employ the same strategy as before, creating the orientation top-down from the clique C_s: For the current C, choose a random ordering of its vertices consistent with the current constraints (the given orientation within its separator set, and the required location of its source and sink). Notice that in cases 1a and 1b, the summations are equivalent, and therefore, each $w' \in \operatorname{Sep}(C_d)$ has an equal chance of being the sink within the orientation restricted to $G[\operatorname{Sep}(C_d)]$. Therefore, the orientation within $\operatorname{Sep}(C_d)$ can be chosen uniformly at random, as long as it is consistent with the orientation of $G[\operatorname{Sep}(C)]$. Thus, the running time of the sampling algorithm is $O(|E(G)|)$. □

4.4 Counting Sink-free Orientations

In this section, we present a polynomial-time exact counter for sink-free orientations via dynamic programming over the clique tree of the given chordal graph. Our algorithm significantly improves the running time over the existing FPRAS for this graph class. We note that there is an efficient uniform sampler for sink-free orientations in any graph. So, we are only focusing on the counting.

4.4.1 Related works

Bubley and Dyer [15] proved that counting of sink-free orientations is #P-complete on general graphs. They also proposed a Markov Chain that samples sink-free orientations of an arbitrary input graph G approximately from the uniform distribution in time $O(|E(G)|^3 \log \epsilon^{-1})$, where ϵ is the degree of approximation. Additionally, they showed that the problem of counting sink-free orientations is self-reducible, yielding a fully polynomial randomized approximation scheme (FPRAS) for the counting problem, the running time of which is roughly $|E(G)|$ times the sampling running time. Huber [31] used the "coupling from the past" technique to obtain an exact sample in time $O(|E(G)|^4)$. Cohn, Pemantle, and Propp [17] proposed a "sink-popping" algorithm which can generate a sink-free orientation uniformly at random in $O(|V(G)||E(G)|)$ time. This algorithm fits the "partial rejection sampling through the Lovász Local Lemma" framework of Guo, Jerrum and Liu [27], yielding a uniformly random sink-free orientation in time $O(|V(G)|^2)$ time.

4.4.2 Our contribution

In this section, we prove the following theorem:

Theorem 17. *Let G be a connected chordal graph. The number of sink-free orientations of G can be counted in $\tilde{O}(|V(G)|) + O(|E(G)|)$ time.*

We note that while our proof of the theorem employs dynamic programming over a clique tree, unlike before, here we need to use parts of an inclusion-exclusion principle to derive our quantities. Counting algorithms based on dynamic programming can often be used to sample: If the algorithm is based on summing counts corresponding to disjoint subproblems, one first runs the counting algorithm, followed by the sampling which proceeds top-down, always choosing which subproblem to go into proportionally to its count. However, here we are subtracting quantities as part of our computations and, as such, a sampling algorithm does not seem to follow from the counting algorithm.

Theorem 17 does yield an exact sampler via the problem's established self-reducibility property [15]. However, recall that there are more efficient exact samplers already known [17], so we do not discuss the sampling aspect here. Our main contribution related to sink-free orientations is an efficient counting algorithm for chordal graphs.

Proof of Theorem 17. Let T be a clique tree of G, rooted at an arbitrary clique C_r. We show how to compute the following quantities for each clique C in T:

- $\mathrm{SFO}(T_C)$: The number of orientations of the graph $\hat{G}[T_C]$ that have no sinks in $V(\hat{G}[T_C]) - \mathrm{Sep}(C)$ (only sinks in $\mathrm{Sep}(C)$ are allowed).
- $\mathrm{ASFO}(T_C, v)$: The number of orientations of the graph $\hat{G}[T_C]$, where $v \in \mathrm{Sep}(C)$ is a sink and there are no sinks in $V(\hat{G}[T_C]) - \mathrm{Sep}(C)$.

Then, the number of sink-free orientations of G is exactly $\mathrm{SFO}(T_{C_r})$, since $\mathrm{Sep}(C_r) = \emptyset$.

Before we proceed with the computation, in order to simplify our expressions we define these additional quantities. For a clique C:

(i) Let $\mathrm{oa}(C)$ be the number of all orientations of the clique C, i.e., $\mathrm{oa}(C) = 2^{\binom{|C|}{2}}$.

(ii) Let $\mathrm{os}(C, v)$ be the number of orientations of the clique C where the vertex $v \in C$ is a sink. All edges have to be oriented towards v, hence $\mathrm{os}(C, v) = 2^{\binom{|C|-1}{2}}$. It also follows that v is the only sink in C. Moreover, since the quantity $\mathrm{os}(C, v)$ does not depend on the vertex v, we may simplify the notation to just $\mathrm{os}(C)$.

(iii) Let $\mathrm{ox}(C)$ be the number of orientations of C where no vertex is a sink. Since each orientation with a sink has a unique sink, see (ii), we get that $\mathrm{ox}(C) = \mathrm{oa}(C) - |C| \,\mathrm{os}(C)$.

We will compute the quantities $\mathrm{SFO}(T_C)$ and $\mathrm{ASFO}(T_C)$ by dynamic programming on the clique tree T_{C_r}.

Base case. Let C be a leaf of T. Recall that $\mathrm{SFO}(T_C)$ and $\mathrm{ASFO}(T_C, v)$ consider only the edges within $\mathrm{Res}(C)$ and those between $\mathrm{Res}(C)$ and $\mathrm{Sep}(C)$. Therefore,

$$\mathrm{ASFO}(T_C, v) = \mathrm{oa}(\mathrm{Res}(C))2^{|\,\mathrm{Res}(C)|(|\,\mathrm{Sep}(C)|-1)},$$

since all edges from $\mathrm{Res}(C)$ have to be oriented towards v, which prevents any vertex in $\mathrm{Res}(C)$ from being a sink; this means that edges within $\mathrm{Res}(C)$, as well as between $\mathrm{Res}(C)$ and $\mathrm{Sep}(C) - \{v\}$ can be oriented arbitrarily.

For $\mathrm{SFO}(T_C)$, we have several mutually exclusive possibilities:

1. The orientation within $\mathrm{Res}(C)$ contains no sinks. Then, the edges between $\mathrm{Res}(C)$ and $\mathrm{Sep}(C)$ can be oriented arbitrarily, getting overall $\mathrm{ox}(\mathrm{Res}(C))2^{|\,\mathrm{Res}(C)||\,\mathrm{Sep}(C)|}$ of such orientations.

2. The orientation within $\mathrm{Res}(C)$ contains a (single) sink $u \in \mathrm{Res}(C)$. Then, at least one of the edges from $\mathrm{Sep}(C)$ needs to be oriented away from u (thus preventing u from remaining as a sink), the other edges between $\mathrm{Sep}(C)$ and $\mathrm{Res}(C) - \{u\}$ can be oriented arbitrarily. This corresponds to $\mathrm{os}(\mathrm{Res}(C), u)(2^{|\,\mathrm{Sep}(C)|} - 1)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}$ of such orientations. (The second factor comes from subtracting the one orientation where all edges from $\mathrm{Sep}(C)$ are oriented towards u.)

Therefore, summing across possible $u \in \mathrm{Res}(C)$, we get

$$\begin{aligned}\mathrm{SFO}(T_C) \;=\; & \mathrm{ox}(\mathrm{Res}(C))2^{|\,\mathrm{Res}(C)||\,\mathrm{Sep}(C)|} + \\ & |\,\mathrm{Res}(C)|\,\mathrm{os}(\mathrm{Res}(C))(2^{|\,\mathrm{Sep}(C)|} - 1)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}.\end{aligned}$$

Inductive case. Let C be a non-leaf of T, and let $C_1, C_2, \ldots, C_d$ be its children cliques in T. We first describe how to compute $\mathrm{ASFO}(T_C, v)$. The same logic for orienting the edges between $\mathrm{Sep}(C)$

and $\mathrm{Res}(C)$ as in the base case applies here: edges towards v are forced, the others are arbitrary. Moreover, just like in the base case, if the edges within $\mathrm{Res}(C)$ are oriented arbitrarily, they will not create a sink in $\mathrm{Res}(C)$ since there is always an edge towards v. We still need to orient the edges in the graphs induced by the subtrees rooted at $C_1, \ldots, C_d$. In particular, we need to orient the graphs $\hat{G}[T_{C_i}]$ for $i = 1, \ldots, d$. We already have the guarantee that when adding the orientation of the edges within C, there will be no sinks in $\mathrm{Res}(C)$. Therefore, we simply need to orient $\hat{G}[T_{C_i}]$ so that there are no sinks outside $\mathrm{Sep}(C_i)$. By Lemma 10, each $\hat{G}[T_{C_i}]$ can be oriented independently. Therefore,

$$\mathrm{ASFO}(T_C, v) = \mathrm{oa}(\mathrm{Res}(C))2^{|\,\mathrm{Res}(C)|(|\,\mathrm{Sep}(C)|-1)} \prod_{i=1}^{d} \mathrm{SFO}(T_{C_i}).$$

It remains to compute $\mathrm{SFO}(T_C)$. The possible orientations can be partitioned into the following mutually exclusive possibilities:

1. The orientation within $\mathrm{Res}(C)$ contains no sinks. Then, the edges between $\mathrm{Res}(C)$ and $\mathrm{Sep}(C)$ can be oriented arbitrarily, and by Lemma 10, the orientations of each $\hat{G}[T_{C_i}]$ are independent and need to contain no sinks outside $\mathrm{Sep}(C_i)$. There are $\mathrm{ox}(\mathrm{Res}(C))2^{|\,\mathrm{Res}(C)||\,\mathrm{Sep}(C)|} \prod_{i=1}^{d} \mathrm{SFO}(T_{C_i})$ of such orientations.

2. The orientation within $\mathrm{Res}(C)$ contains a (single) sink $u \in \mathrm{Res}(C)$. We have two subcases:

   a) There exists an edge from u to $\mathrm{Sep}(C)$ oriented away from u (and, therefore, u cannot be a sink). In this case, the orientations of $\hat{G}[T_{C_i}]$ can be arbitrary as long as there are no sinks outside $\mathrm{Sep}(C_i)$. There are $\mathrm{os}(\mathrm{Res}(C), u)(2^{|\,\mathrm{Sep}(C)|} - 1)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)} \prod_{i=1}^{d} \mathrm{SFO}(T_{C_i})$ of such orientations.

   b) All edges from u to $\mathrm{Sep}(C)$ are oriented towards u. Since $\mathrm{SFO}(T_C)$ counts orientations with no sinks outside $\mathrm{Sep}(C)$, these partial orientations need to be completed within the $\hat{G}[T_{C_i}]$ graphs in a way that ensures that u is not a sink. The vertex u remains a sink if all edges within each $\hat{G}[T_{C_i}]$ point towards u. In other words, these are exactly the orientations counted in $\mathrm{ASFO}(C_i, u)$. We can subtract these orientations from the total number. In particular, let $I_u \subseteq \{1, \ldots, d\}$ be the indices for which $u \in \mathrm{Sep}(C_i)$. Then, the number of orientations of $\hat{G}[T_C]$ in this subcase is

$$\mathrm{os}(\mathrm{Res}(C), u)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)} \prod_{i \in [d]-I_u} \mathrm{SFO}(T_{C_i}) \left(\prod_{i \in I_u} \mathrm{SFO}(T_{C_i}) - \prod_{i \in I_u} \mathrm{ASFO}(T_{C_i}, u) \right).$$

Putting it all together we get

$$\begin{aligned}\mathrm{SFO}(T_C) &= \mathrm{ox}(\mathrm{Res}(C))2^{|\mathrm{Res}(C)||\mathrm{Sep}(C)|}\prod_{i=1}^{d}\mathrm{SFO}(T_{C_i}) + \\ &\sum_{u\in\mathrm{Res}(C)}\mathrm{os}(\mathrm{Res}(C),u)(2^{|\mathrm{Sep}(C)|}-1)2^{|\mathrm{Sep}(C)|(|\mathrm{Res}(C)|-1)}\prod_{i=1}^{d}\mathrm{SFO}(T_{C_i}) + \\ &\sum_{u\in\mathrm{Res}(C)}\mathrm{os}(\mathrm{Res}(C),u)2^{|\mathrm{Sep}(C)|(|\mathrm{Res}(C)|-1)}\prod_{i\in[d]-I_u}\mathrm{SFO}(T_{C_i})\times \\ &\left(\prod_{i\in I_u}\mathrm{SFO}(T_{C_i})-\prod_{i\in I_u}\mathrm{ASFO}(T_{C_i},u)\right).\end{aligned}$$

Finally, we need to estimate the running time of the algorithm. Notice that the size of the set I_u corresponding to the children cliques of C_u that contain u, is upper-bounded by $\deg_G(u)$. This is because for each child clique C_i for $i \in I_u$ we have a $w_i \in \mathrm{Res}(C_i)$. All the w_i's are distinct since $\mathrm{Sep}(C)$ separates them, yielding $|I_u| \leq \deg_G(u)$.

Next notice that we do not need to store the quantities $\mathrm{ASFO}(T_C, v)$ for each $v \in \mathrm{Sep}(C)$. This is because the calculation of $\mathrm{ASFO}(T_C, v)$ is independent of v and therefore we really need only one quantity $\mathrm{ASFO}(T_C)$ for each clique.

The base case of the algorithm takes $O(1)$ arithmetic operations per leaf clique. For the inductive case, the computation of the $\mathrm{ASFO}(T_C)$ takes $O(d)$ arithmetic operations, assuming $\mathrm{SFO}(T_{C_i})$ has been computed. Noticing that $d = \mathrm{outdeg}_T(C)$, the computation of all ASFO's across the entire tree T takes $O(\sum_{C\in T}\mathrm{outdeg}_T(C)) = O(|T|) = O(|V(G)|)$ arithmetic operations, if the SFO's of the children are available. We can rewrite the computation of the $\mathrm{SFO}(T_C)$ as follows:

$$\begin{aligned}\mathrm{SFO}(T_C) &= \prod_{i=1}^{d}\mathrm{SFO}(T_{C_i})[\mathrm{ox}(\mathrm{Res}(C))2^{|\mathrm{Res}(C)||\mathrm{Sep}(C)|} + \\ &|\mathrm{Res}(C)|\,\mathrm{os}(\mathrm{Res}(C))(2^{|\mathrm{Sep}(C)|}-1)2^{|\mathrm{Sep}(C)|(|\mathrm{Res}(C)|-1)} + \\ &\sum_{u\in\mathrm{Res}(C)}\left(1-\prod_{i\in I_u}\frac{\mathrm{ASFO}(T_{C_i},u)}{\mathrm{SFO}(T_{C_i})}\right)]\end{aligned}$$

The computation of $\prod_{i=1}^{d}\mathrm{SFO}(T_{C_i})$ has already been accounted for during the computation of ASFO. Therefore, we only focus on the terms in the square brackets. The first two terms take $O(1)$ arithmetic operations. The last term takes $O(\sum_{u\in\mathrm{Res}(C)}|I_u|)$ operations. Since $|I_u| \leq \deg_G(u)$,

across the entire tree T we get $O(\sum_{C\in T}\sum_{u\in \mathrm{Res}(C)} \deg_G(u)) = O(\sum_{u\in V(G)} \deg_G(u)) = O(|E(G)|)$ operations. This concludes the proof of the theorem. □

4.5 Counting Source-Sink-free Orientations

A source-sink-free orientation does not have any sources nor sinks. A strong orientation is a source-sink-free orientation that has only one strongly connected component. Counting strong orientations of a given graph G is #P-complete [81], and it is also #P-complete when G is planar and bipartite [78]. Alon [3] gave an FPRAS for strong orientations in a dense graph if its minimum degree is linear in the number of vertices. To the best of our knowledge, the hardness of #strong-orientations over chordal graphs is unknown. In this section, we present a polynomial-time exact counter for the more general source-sink-free orientations on chordal graphs. Our main motivation for studying these orientations is that, despite their similarity to the sink-free orientations, where there are several successful sampling approaches that yield the corresponding FPRASs for counting, these sampling approaches do not seem to extend to source-sink-free orientations. Consequently, prior to our work, there were no known results related to the counting of these orientations.

4.5.1 Our contributions

Our counting approach for source-sink-free orientations follows a similar outline to the approach for sink-free orientations; however, there is a complication involving a more complex inclusion-exclusion principle, which was not present in the earlier problem. We define the following quantities for each clique C in a rooted clique tree T of the given chordal graph G:

- SSFO(T_C): The number of orientations of the graph $\hat{G}[T_C]$ where every sink and every source is in Sep(C). Let $\mathcal{S}(T_C)$ be the set of all these orientations.
- SoO(T_C, v_1): The number of orientations in $\mathcal{S}(T_C)$, where $v_1 \in \mathrm{Sep}(C)$ is a source.
- SiO(T_C, v_2): The number of orientations in $\mathcal{S}(T_C)$, where $v_2 \in \mathrm{Sep}(C)$ is a sink.
- SoSiO(T_C, v_1, v_2): The number of orientations in $\mathcal{S}(T_C)$, where $v_1 \in \mathrm{Sep}(C)$ is a source and $v_2 \in \mathrm{Sep}(C)$ is a sink. Notice that since $\mathrm{Res}(C) \neq \emptyset$, it follows that $v_1 \neq v_2$.

The quantity $\mathrm{SSFO}(T_{C_r})$ computes the number of source-sink-free orientations of G, where C_r is the root clique of the clique tree T.

In addition to the quantities $\mathrm{oa}(C)$, $\mathrm{os}(C, v)$, and $\mathrm{ox}(C)$ defined for a given clique C in Section 4.4.2, here we will use also these additional quantities:

(i) Let $\mathrm{oss}(C, v_1, v_2)$ be the number of orientations of C where v_1 is a source and v_2 is a sink. It follows that $\mathrm{oss}(C, v_1, v_2) = 2^{\binom{|C|-2}{2}}$. Since the value does not depend on v_1, v_2, we simplify the notation to just $\mathrm{oss}(C)$.

(ii) Let $\mathrm{oxx}(C)$ be the number of orientations of C with no sources or sinks. An oriented clique can have at most one source and at most one sink. Therefore, from all orientations we can subtract those that have a sink and those that have a source; leading to "double penalization" of orientations with both a source and a sink. Therefore, $\mathrm{oxx}(C) = \mathrm{oa}(C) - 2|C|\,\mathrm{os}(C) + \binom{|C|}{2}\mathrm{oss}(C)$.

As before, we will compute the quantities $\mathrm{SSFO}(T_C)$, $\mathrm{SoO}(T_C, v_1)$, $\mathrm{SiO}(T_C, v_2)$, and $\mathrm{SoSiO}(T_C, v_1, v_2)$ by dynamic programming on the rooted clique tree T.

Base case. Let C be a leaf of T. In $\mathrm{SoO}(T_C, v_1)$ and $\mathrm{SoSiO}(T_C, v_1, v_2)$ all edges incident to v_1 point away from v_1, and in $\mathrm{SiO}(T_C, v_2)$ and $\mathrm{SoSiO}(T_C, v_1, v_2)$ the edges incident to v_2 need to point towards v_2; the other edges can be oriented either way. We get:

$$\mathrm{SoO}(T_C, v_1) = \mathrm{SiO}(T_C, v_2) = \mathrm{oa}(\mathrm{Res}(C))2^{|\mathrm{Res}(C)|(|\mathrm{Sep}(C)|-1)},$$

$$\mathrm{SoSiO}(T_C, v_1, v_2) = \mathrm{oa}(\mathrm{Res}(C))2^{|\mathrm{Res}(C)|(|\mathrm{Sep}(C)|-2)}.$$

For $\mathrm{SSFO}(T_C)$, we partition $\mathcal{S}(T_C)$ into these mutually exclusive possibilities.

1. The orientation restricted to $G[\mathrm{Res}(C)]$ contains no sources nor sinks. Then, the edges between $\mathrm{Res}(C)$ and $\mathrm{Sep}(C)$ can be oriented arbitrarily, leading to $\mathrm{oxx}(\mathrm{Res}(C))2^{|\mathrm{Res}(C)||\mathrm{Sep}(C)|}$ of such orientations.

2. The orientation restricted to $G[\mathrm{Res}(C)]$ contains a (single) source $u_1 \in \mathrm{Res}(C)$ and no sinks. Then, at least one of the edges from $\mathrm{Sep}(C)$ needs to be oriented towards u_1 to prevent it from remaining a source, and the other edges between $\mathrm{Sep}(C)$ and $\mathrm{Res}(C) - \{u_1\}$ can be oriented arbitrarily. The part of the orientation within $\mathrm{Res}(C)$ has to have all edges outgoing from u_1, and the remaining edges must be oriented so that there is no sink within $\mathrm{Res}(C) - \{u_1\}$. This corresponds to $\mathrm{ox}(\mathrm{Res}(C) - \{u_1\})(2^{|\mathrm{Sep}(C)|} - 1)2^{|\mathrm{Sep}(C)|(|\mathrm{Res}(C)|-1)}$ of such orientations.

3. The orientation restricted to $G[\mathrm{Res}(C)]$ contains a (single) sink $u_2 \in \mathrm{Res}(C)$ and no sources. The calculation is analogous to the previous case.

4. The orientation restricted to $G[\mathrm{Res}(C)]$ contains a (single) source u_1 and a (single) sink u_2. Then, u_1 needs to be "fixed" by at least one edge from $\mathrm{Sep}(C)$, u_2 by at least one edge to $\mathrm{Sep}(C)$, and the other edges between $\mathrm{Sep}(C)$ and $\mathrm{Res}(C)$ can be oriented arbitrarily. Likewise, the edges within $\mathrm{Res}(C) - \{u_1, u_2\}$ can be oriented arbitrarily. We get $\mathrm{oa}(\mathrm{Res}(C) - \{u_1, u_2\})(2^{|\,\mathrm{Sep}(C)|} - 1)^2 2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-2)}$ of such orientations.

Therefore, summing across possible $u_1, u_2 \in \mathrm{Res}(C)$, we get

$$\begin{aligned}\mathrm{SSFO}(T_C) = {} & \mathrm{oxx}(\mathrm{Res}(C))2^{|\,\mathrm{Res}(C)||\,\mathrm{Sep}(C)|} + \\ & 2|\,\mathrm{Res}(C)|\,\mathrm{ox}(\mathrm{Res}(C)^{-1})(2^{|\,\mathrm{Sep}(C)|} - 1)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)} + \\ & \binom{|\,\mathrm{Res}(C)|}{2}\,\mathrm{oa}(\mathrm{Res}(C)^{-2})(2^{|\,\mathrm{Sep}(C)|} - 1)^2 2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-2)},\end{aligned}$$

where for a clique $\hat{C}$, the notation $\hat{C}^{-k}$ stands for removing k vertices from $\hat{C}$.

Inductive case. Let C be a non-leaf of T, and let $C_1, C_2, \ldots, C_d$ be its children cliques in T. As before, for $u \in \mathrm{Res}(C)$ we denote by I_u the set of indices such that $u \in C_i$ for $i \in I_u$.

To compute $\mathrm{SoSiO}(T_C, v_1, v_2)$, the edges between v_1, respectively v_2, and $\mathrm{Res}(C)$ are forced (away from v_1, towards v_2). This implies that no vertex in $\mathrm{Res}(C)$ will be a source, nor sink, and hence the orientation of all other edges can be arbitrary. We get:

$$\mathrm{SoSiO}(T_C, v_1, v_2) = \mathrm{oa}(\mathrm{Res}(C))2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-2)} \prod_{i=1}^{d} \mathrm{SSFO}(T_{C_i}).$$

For $\mathrm{SoO}(T_C, v_1)$, the edges between v_1 and $\mathrm{Res}(C)$ need to point away from v_1, and as such there will be no sources in $\mathrm{Res}(C)$. We distinguish two cases:

1. There is no sink in the orientation restricted to $G[\mathrm{Res}(C)]$. The number of orientations of $\hat{G}[T_C]$ corresponding to this case is

$$\mathrm{ox}(\mathrm{Res}(C))2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)} \prod_{i=1}^{d} \mathrm{SSFO}(T_{C_i}).$$

2. The orientation restricted to $G[\mathrm{Res}(C)]$ contains a (single) sink u_2. Then either there is an edge between u_2 and $\mathrm{Sep}(C)$ pointing towards u_2, or all edges point away from u_2 and u_2 cannot be

a sink at at least one of the children subtrees. We get

$$\begin{aligned}&\mathrm{os}(\mathrm{Res}(C), u_2)(2^{|\,\mathrm{Sep}(C)|}-1)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}\prod_{i=1}^{d}\mathrm{SSFO}(T_{C_i})+\\&\mathrm{os}(\mathrm{Res}(C), u_2)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}\prod_{i\in[d]-I_{u_2}}\mathrm{SSFO}(T_{C_i})\times\\&\left(\prod_{i\in I_{u_2}}\mathrm{SSFO}(T_{C_i})-\prod_{i\in I_{u_2}}\mathrm{SiO}(T_{C_i}, u_2)\right).\end{aligned}$$

orientations of $\hat{G}[T_C]$ corresponding to this case.

Therefore, $\mathrm{SoO}(T_C, v_1)$ can be computed as follows:

$$\begin{aligned}\mathrm{SoO}(T_C, v_1) = {}&\mathrm{ox}(\mathrm{Res}(C))2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}\prod_{i=1}^{d}\mathrm{SSFO}(T_{C_i})+\\&\sum_{u_2\in\mathrm{Res}(C)}\mathrm{os}(\mathrm{Res}(C), u_2)(2^{|\,\mathrm{Sep}(C)|}-1)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}\prod_{i=1}^{d}\mathrm{SSFO}(T_{C_i})+\\&\sum_{u_2\in\mathrm{Res}(C)}\mathrm{os}(\mathrm{Res}(C), u_2)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}\prod_{i\in[d]-I_{u_2}}\mathrm{SSFO}(T_{C_i})\times\\&\left(\prod_{i\in I_{u_2}}\mathrm{SSFO}(T_{C_i})-\prod_{i\in I_{u_2}}\mathrm{SiO}(T_{C_i}, u_2)\right).\end{aligned}$$

Analogously, we get

$$\begin{aligned}\mathrm{SiO}(T_C, v_1) = {}&\mathrm{ox}(\mathrm{Res}(C))2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}\prod_{i=1}^{d}\mathrm{SSFO}(T_{C_i})+\\&\sum_{u_1\in\mathrm{Res}(C)}\mathrm{os}(\mathrm{Res}(C), u_1)(2^{|\,\mathrm{Sep}(C)|}-1)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}\prod_{i=1}^{d}\mathrm{SSFO}(T_{C_i})+\\&\sum_{u_1\in\mathrm{Res}(C)}\mathrm{os}(\mathrm{Res}(C), u_1)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}\prod_{i\in[d]-I_{u_1}}\mathrm{SSFO}(T_{C_i})\times\\&\left(\prod_{i\in I_{u_1}}\mathrm{SSFO}(T_{C_i})-\prod_{i\in I_{u_1}}\mathrm{SoO}(T_{C_i}, u_1)\right).\end{aligned}$$

Finally, we need to compute $\mathrm{SSFO}(T_C)$. We will split the possible orientations into these four mutually exclusive cases:

1. The orientation restricted to $G[\mathrm{Res}(C)]$ contains no sources nor sinks. Then, all the remaining edges with $\mathrm{Res}(C)$ can be oriented arbitrarily, and the children subtrees can be oriented recursively (provided, as always, that there are no sinks or sources outside their separator sets). Therefore, the number of these orientations is $\mathrm{ox}(\mathrm{Res}(C))2^{|\mathrm{Sep}(C)||\mathrm{Res}(C)|}\prod_{i=1}^{d}\mathrm{SSFO}(T_{C_i})$.

2. The orientation restricted to $G[\mathrm{Res}(C)]$ contains a (single) source u_1 and no sinks. Then, either there is an edge oriented from $\mathrm{Sep}(C)$ to u_1, or all edges are oriented from u_1 to $\mathrm{Sep}(C)$ and one of the children subtrees does not have u_1 as their source. Within $\mathrm{Res}(C)$, all edges point away from u_1 and the remainder of $\mathrm{Res}(C)$ needs to be sink-free. Thus, the number of orientations of $\hat{G}[T_C]$ corresponding to this case is:

$$\begin{aligned}&\mathrm{ox}(\mathrm{Res}(C)^{-1})(2^{|\mathrm{Sep}(C)|}-1)2^{|\mathrm{Sep}(C)|(|\mathrm{Res}(C)|-1)}\prod_{i=1}^{d}\mathrm{SSFO}(T_{C_i})+\\&\mathrm{ox}(\mathrm{Res}(C)^{-1})2^{|\mathrm{Sep}(C)|(|\mathrm{Res}(C)|-1)}\prod_{i\in[d]-I_{u_1}}\mathrm{SSFO}(T_{C_i})\times\\&\left(\prod_{i\in I_{u_1}}\mathrm{SSFO}(T_{C_i})-\prod_{i\in I_{u_1}}\mathrm{SoO}(T_{C_i},u_1)\right).\end{aligned}$$

3. The orientation restricted to $G[\mathrm{Res}(C)]$ contains a (single) sink u_2 and no sources. Analogously to the previous case, the number of the corresponding orientations of $\hat{G}[T_C]$ is:

$$\begin{aligned}&\mathrm{ox}(\mathrm{Res}(C)^{-1})(2^{|\mathrm{Sep}(C)|}-1)2^{|\mathrm{Sep}(C)|(|\mathrm{Res}(C)|-1)}\prod_{i=1}^{d}\mathrm{SSFO}(T_{C_i})+\\&\mathrm{ox}(\mathrm{Res}(C)^{-1})2^{|\mathrm{Sep}(C)|(|\mathrm{Res}(C)|-1)}\prod_{i\in[d]-I_{u_2}}\mathrm{SSFO}(T_{C_i})\times\\&\left(\prod_{i\in I_{u_2}}\mathrm{SSFO}(T_{C_i})-\prod_{i\in I_{u_2}}\mathrm{SiO}(T_{C_i},u_2)\right).\end{aligned}$$

4. The orientation restricted to $G[\mathrm{Res}(C)]$ contains a (single) source u_1 and a (single) sink u_2. We will partition the corresponding orientations of $\hat{G}[T_C]$ into these subcases:

   a) There is an edge from $\mathrm{Sep}(C)$ to u_1 and from u_2 to $\mathrm{Sep}(C)$. (I.e., both u_1 and u_2 are "fixed" by an edge from/to $\mathrm{Sep}(C)$.) The number of corresponding orientations is

$$\mathrm{oss}(\mathrm{Res}(C))(2^{|\mathrm{Sep}(C)|}-1)^2 2^{|\mathrm{Sep}(C)|(|\mathrm{Res}(C)|-2)}\prod_{i=1}^{d}\mathrm{SSFO}(T_{C_i}).$$

b) There is an edge from $\mathrm{Sep}(C)$ to u_1 but no edge from u_2 to $\mathrm{Sep}(C)$. (I.e., u_1 is "fixed" by $\mathrm{Sep}(C)$ but u_2 is not.) Then u_2 needs to be "fixed" by one of the children subtrees. The number of corresponding orientations is

$$\mathrm{oss}(\mathrm{Res}(C))(2^{|\,\mathrm{Sep}(C)|}-1)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-2)}\prod_{i\in[d]-I_{u_2}}\mathrm{SSFO}(T_{C_i})\times$$
$$\left(\prod_{i\in I_{u_2}}\mathrm{SSFO}(T_{C_i})-\prod_{i\in I_{u_2}}\mathrm{SiO}(T_{C_i},u_2)\right).$$

c) There is an edge from u_2 to $\mathrm{Sep}(C)$ but no edge from $\mathrm{Sep}(C)$ to u_1. (I.e., u_2 is "fixed" by $\mathrm{Sep}(C)$ but u_1 is not.) Analogous to the previous subcase, the number of corresponding orientations is

$$\mathrm{oss}(\mathrm{Res}(C))(2^{|\,\mathrm{Sep}(C)|}-1)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-2)}\prod_{i\in[d]-I_{u_1}}\mathrm{SSFO}(T_{C_i})\times$$
$$\left(\prod_{i\in I_{u_1}}\mathrm{SSFO}(T_{C_i})-\prod_{i\in I_{u_1}}\mathrm{SoO}(T_{C_i},u_1)\right).$$

d) There is no edge from $\mathrm{Sep}(C)$ to u_1 and no edge from u_2 to $\mathrm{Sep}(C)$. (I.e., neither u_1 nor u_2 is "fixed" by $\mathrm{Sep}(C)$.) Then both u_1 and u_2 need to be "fixed" by one of the children subtrees. Let X_{u_1} be the set of valid orientations of the subtrees where no subtree fixes u_1. (We call an orientation of the subtrees valid if sinks and sources are present only in the residual sets in the root cliques of each tree.) Then,

$$|X_{u_1}|=\prod_{i\in[d]-I_{u_1}}\mathrm{SSFO}(T_{C_i})\prod_{i\in I_{u_1}}\mathrm{SoO}(T_{C_i},u_1).$$

Let Y_{u_1} be the set of valid orientations of the subtrees where no subtree fixes u_2. Then,

$$|Y_{u_2}|=\prod_{i\in[d]-I_{u_2}}\mathrm{SSFO}(T_{C_i})\prod_{i\in I_{u_2}}\mathrm{SiO}(T_{C_i},u_2).$$

Let $Z_{u_1,u_2}=X_{u_1}\cap Y_{u_2}$. In particular, Z_{u_1,u_2} is the set of all valid orientations of the subtrees where no subtree fixes u_1 nor u_2. In other words, the subtrees containing u_1 but not u_2 have u_1 as a source, the subtrees containing u_2 but not u_1 have u_2 as a sink, and the subtrees containing both u_1 and u_2 have u_1 as a source and u_2 as a sink. Therefore,

$$|Z_{u_1,u_2}|=\prod_{i\in[d]-I_{u_1}-I_{u_2}}\mathrm{SSFO}(T_{C_i})\prod_{i\in I_{u_1}-I_{u_2}}\mathrm{SoO}(T_{C_i},u_1)\times$$
$$\prod_{i\in I_{u_2}-I_{u_1}}\mathrm{SiO}(T_{C_i},u_2)\prod_{i\in I_{u_1}\cap I_{u_2}}\mathrm{SoSiO}(T_{C_i},u_1,u_2).$$

Then, when accounting for all orientations in case 4d, we use inclusion-exclusion as follows: We consider all valid orientations of the subtrees, then subtract those in X_{u_1} and Y_{u_2}, and then add those in Z_{u_1,u_2} to compensate for double subtraction. Therefore, the number of orientations of $\hat{G}[T_C]$ corresponding to case 4d is

$$\operatorname{oss}(\operatorname{Res}(C))2^{|\operatorname{Sep}(C)|(|\operatorname{Res}(C)|-2)}\left(\prod_{i\in[d]}\operatorname{SSFO}(T_{C_i})-|X_{u_1}|-|Y_{u_2}|+|Z_{u_1,u_2}|\right).$$

Putting it all together we get

$$\begin{aligned}
\operatorname{SSFO}(T_C) = {} & \operatorname{ox}(\operatorname{Res}(C))2^{|\operatorname{Sep}(C)||\operatorname{Res}(C)|}\prod_{i=1}^{d}\operatorname{SSFO}(T_{C_i})+ \\
& \sum_{u_1\in\operatorname{Res}(C)}\operatorname{ox}(\operatorname{Res}(C)^{-1})2^{|\operatorname{Sep}(C)|(|\operatorname{Res}(C)|-1)}\prod_{i=1}^{d}\operatorname{SSFO}(T_{C_i})\times \\
& \left(2^{|\operatorname{Sep}(C)|}-\prod_{i\in I_{u_1}}\frac{\operatorname{SoO}(T_{C_i},u_1)}{\operatorname{SSFO}(T_{C_i})}\right)+ \\
& \sum_{u_2\in\operatorname{Res}(C)}\operatorname{ox}(\operatorname{Res}(C)^{-1})2^{|\operatorname{Sep}(C)|(|\operatorname{Res}(C)|-1)}\prod_{i=1}^{d}\operatorname{SSFO}(T_{C_i})\times \\
& \left(2^{|\operatorname{Sep}(C)|}-\prod_{i\in I_{u_2}}\frac{\operatorname{SiO}(T_{C_i},u_2)}{\operatorname{SSFO}(T_{C_i})}\right)+ \\
& \sum_{u_1,u_2\in\operatorname{Res}(C),u_1\neq u_2}\operatorname{oss}(\operatorname{Res}(C))2^{|\operatorname{Sep}(C)|(|\operatorname{Res}(C)|-2)}\prod_{i=1}^{d}\operatorname{SSFO}(T_{C_i})\times \\
& \left[(2^{|\operatorname{Sep}(C)|}-1)^2+(2^{|\operatorname{Sep}(C)|}-1)\left(1-\prod_{i\in I_{u_2}}\frac{\operatorname{SiO}(T_{C_i},u_2)}{\operatorname{SSFO}(T_{C_i})}\right)+\right. \\
& \left.(2^{|\operatorname{Sep}(C)|}-1)\left(1-\prod_{i\in I_{u_1}}\frac{\operatorname{SoO}(T_{C_i},u_1)}{\operatorname{SSFO}(T_{C_i})}\right)\right]+ \\
& \sum_{u_1,u_2\in\operatorname{Res}(C),u_1\neq u_2}\operatorname{oss}(\operatorname{Res}(C))2^{|\operatorname{Sep}(C)|(|\operatorname{Res}(C)|-2)}\times \\
& \left(\prod_{i\in[d]}\operatorname{SSFO}(T_{C_i})-|X_{u_1}|-|Y_{u_2}|+|Z_{u_1,u_2}|\right).
\end{aligned}$$

Finally, we need to estimate the running time of the algorithm. The base case is $O(1)$ arithmetic operations per leaf clique. For the inductive case, if we were only considering orientations in subcases 1, 2, 3, 4a, 4b, and 4c, then we would get a linear upper bound on the number of needed arithmetic operations, using the same reasoning as in Section 4.4. However, here we also need to account for case 4d. In particular, we want to efficiently compute

$$\sum_{u_1,u_2\in\mathrm{Res}(C),u_1\neq u_2}[\prod_{i\in[d]}\mathrm{SSFO}(T_{C_i})-\prod_{i\in[d]-I_{u_1}}\mathrm{SSFO}(T_{C_i})\prod_{i\in I_{u_1}}\mathrm{SoO}(T_{C_i},u_1)-$$
$$\prod_{i\in[d]-I_{u_2}}\mathrm{SSFO}(T_{C_i})\prod_{i\in I_{u_2}}\mathrm{SiO}(T_{C_i},u_2)+\prod_{i\in[d]-I_{u_1}-I_{u_2}}\mathrm{SSFO}(T_{C_i})\times$$
$$\prod_{i\in I_{u_1}-I_{u_2}}\mathrm{SoO}(T_{C_i},u_1)\prod_{i\in I_{u_2}-I_{u_1}}\mathrm{SiO}(T_{C_i},u_2)\prod_{i\in I_{u_1}\cap I_{u_2}}\mathrm{SoSiO}(T_{C_i},u_1,u_2)].$$

This can be simplified as

$$\prod_{i\in[d]}\mathrm{SSFO}(T_{C_i})\sum_{u_1,u_2\in\mathrm{Res}(C),u_1\neq u_2}[(1-\prod_{i\in I_{u_1}}\frac{\mathrm{SoO}(T_{C_i},u_1)}{\mathrm{SSFO}(T_{C_i})}-\prod_{i\in I_{u_2}}\frac{\mathrm{SiO}(T_{C_i},u_2)}{\mathrm{SSFO}(T_{C_i})}+$$
$$\prod_{i\in I_{u_1}-I_{u_2}}\frac{\mathrm{SoO}(T_{C_i},u_1)}{\mathrm{SSFO}(T_{C_i})}\prod_{i\in I_{u_2}-I_{u_1}}\frac{\mathrm{SiO}(T_{C_i},u_2)}{\mathrm{SSFO}(T_{C_i})}\prod_{i\in I_{u_1}\cap I_{u_2}}\frac{\mathrm{SoSiO}(T_{C_i},u_1,u_2)}{\mathrm{SSFO}(T_{C_i})}].$$

The first three products can be computed within the linear number of arithmetic operations discussed earlier. For the remaining part of the calculation, we get the following bound on the number of arithmetic operations across all cliques in T (recall that $|I_u|\leq\deg_G(u)$):

$$O\left(\sum_{C\in T}\sum_{u_1,u_2\in\mathrm{Res}(C)}[\deg_G(u_1)+\deg_G(u_2)]\right)=$$
$$O\left(\sum_{C\in T}\sum_{u_1,u_2\in\mathrm{Res}(C)}\deg_G(u_1)+\sum_{C\in T}\sum_{u_1,u_2\in\mathrm{Res}(C)}\deg_G(u_2)\right)=$$
$$O\left(\sum_{C\in T}\sum_{u_1\in\mathrm{Res}(C)}|C|\deg_G(u_1)\right)=O\left(\sum_{v\in V(G)}|C_{\max}|\deg_G(v)\right)=O\left(|C_{\max}||E(G)|\right).$$

The above discussion yields the following theorem.

Theorem 18. *Let G be a chordal graph. The number of source-sink-free orientations of G can be computed in time $\tilde{O}(|C_{\max}||E(G)|)=\tilde{O}(|E(G)||V(G)|)$, where $C_{\max}$ is a maximum clique of G.*

Counting algorithms based on dynamic programming can often be used to sample: If the algorithm is based on summing counts corresponding to disjoint subproblems, one first runs the counting

algorithm, followed by the sampling which proceeds top-down, always choosing which subproblem to go into proportionally to its count. However, here we are subtracting quantities as part of our computations and, as such, a sampling algorithm does not seem to follow from the counting algorithm. For a single level inclusion-exclusion (a single subtraction), one could employ rejection sampling to reject the unfavorable (i.e., those that are subtracted) configurations. However, if almost all configurations are rejected, the probability of sampling success could be minuscule. For two-level inclusion-exclusion, as is the case for our algorithm, even this (potentially low-probability and hence large running time) approach is unclear. We leave the problem of efficient (almost) uniform sampling of source-sink-free orientations in chordal graphs open.

4.6 Sampling Partial Acyclic Orientations by the Lovász Local Lemma

For an undirected graph, a partial acyclic orientation is an assignment of directions to a subset of its edges such that the directed edges do not form a directed cycle. We are interested in sampling partial acyclic orientations in polynomial time, with the hope that this will shed light on the possibility of sampling acyclic orientations in general graphs. In this section, we focus on chordal graphs and initiate a study of applying the partial rejection sampling framework [27] to the problem of sampling partial acyclic orientations from a Gibbs-like underlying distribution. In other words, each partial acyclic orientation will be sampled with probability proportional to $\lambda^{\text{number of directed edges}}$, where $\lambda \in \mathbb{R}^+$. This work was published in AAAI '21 [70].

4.6.1 Partial rejection sampling by the Lovász Local Lemma

The Lováz Local Lemma (LLL) is a powerful tool in combinatorics that guarantees the existence of a perfect object that avoids all "bad" events. Moser and Tardos [53] designed algorithms to find a perfect object under conditions that match the LLL in the variable framework (introduced below). Later on Guo, Jerrum, and Liu [27] designed the partial rejection sampling based on this framework and proved under this framework that some classical sampling algorithms, such as the "cycle-popping" algorithm by Propp and Wilson [58] for sampling rooted spanning trees and the "sink-popping" algorithm by Cohn et al. [17] for sampling sink-free orientations, do provide samples uniformly at random.

We will first introduce the "variable" framework before we talk about this sampling technique. Let

$\mathcal{X} = \{X_1, \ldots, X_n\}$ be a set of n random variables and each of which has its own distribution. Let $\mathcal{A} = \{A_1, \ldots, A_m\}$ be a set of m "bad" events that depend on X_i's, we use $\text{var}(A_i)$ to denote the variables that A_i depends on. A dependency graph $\mathcal{G} = (\mathcal{A}, \mathcal{E})$ is a graph that has vertex set $\mathcal{X}$ and edge set $\mathcal{E}$. If (A_i, A_j) are connected in $\mathcal{G}$ if $\text{var}(A_i) \cap \text{var}(A_j) \neq \emptyset$. For example, in the problem of sampling sink-free orientations [15], where each edge of a given undirected graph G needs to be directed and no vertex is allowed to have only incoming edges. A bad event is a sink vertex, and its variables are edges incident to that vertex. The dependency graph for this problem is just the original input graph.

A problem satisfies the so called "extremal condition" if $(A_i, A_j) \in \mathcal{E}$ implies $\Pr(A_i \wedge A_j) = 0$. It means that two bad events is either independent or cannot occur at the same time. For example, the problem of sampling sink-free orientation satisfies the extremal condition because two adjacent vertices cannot be both sinks. Algorithm 2 in [27] simply keeps resampling variables that are involved in bad events in parallel until there is no bad event. When the extremal condition holds, Guo, Jerrum, and Liu proved that once the algorithm halts, the output is a product distribution conditioned on avoiding all bad events. For sink-free orientations, since each edge will be directed in either direction with equal probability, the output sink-free orientation is just from the uniform distribution.

However, not all problems satisfy the extremal condition. To address this, Guo, Jerrum, and Liu [27] extended the framework beyond the extremal condition, i.e., when two dependent bad events can occur at the same time. In addition to resampling the set $\mathcal{X}$ of the variables involved in the current bad events, they also resample the variables for which some values trigger a new bad event with the current assignment of the variables in $\mathcal{X}$. The following is the general sampling algorithm:

Algorithm 1 General partial rejection sampling

Draw independent samples of all variables from their respective distributions.
while there is an bad event under the current assignment **do**
    Let $\mathcal{R}$ be the set of bad events.
    Let $\mathcal{N}$ be the set of events that will not be resampled, be $\emptyset$.
    Let $\partial\mathcal{R}$ denote the boundary of $\mathcal{R}$ (neighbors of $\mathcal{R}$ outside $\mathcal{R}$) in the dependency graph.
    Let $U \leftarrow \partial\mathcal{R}\backslash\mathcal{N}$
    while $U \neq \emptyset$ **do**
        for each event $A_i \in U$ **do**
            if A_i can be extended to occur **then**
                Add A_i to R
            else
                Add A_i to $\mathcal{N}$.
        $U \leftarrow \partial\mathcal{R}\backslash\mathcal{N}$
    for each variable in $\bigcup_{A_i\in\mathcal{R}} \mathrm{var}(A_i)$ **do**
        Resample it according to its own distribution.
Output G.

The resulting distribution is still a product distribution conditioned on no occurring bad events, and the expected running time is stated in the following theorem.

Theorem 19 (Guo-Jerrum-Liu)**.** *Let n be the number of variables, N be the number of bad events, and Δ be the maximum degree of the dependency graph. For any bad event, let p be an upper bound on the probability that this bad event occurs. Finally, let r be the maximum probability such that for a pair of neighboring bad events A, B and an assignment of values to the variables in A, if the variables in $\mathrm{var}(B) \setminus \mathrm{var}(A)$ are drawn, B occurs. Then, for any $\Delta \geq 2$, if $6ep\Delta^2 \leq 1$ and $3er\Delta \leq 1$, the expected number of resampling rounds is $O(\log N)$ and thus the expected number of variable resamples is $O(n \log N)$ with high probability.*

4.6.2 Our contributions

We define a random variable X_e for every edge $e = (u, v)$ of a given chordal graph G. This random variable will take one of three possible values: direction from u to v, direction from v to u, or staying undirected, chosen from distribution $[q, q, 1-2q]$ for some $q \in [0, 1]$ that will be specified soon. We define bad events as directed cycles of length three (triangle cycles). We note that the partial acyclic orientation does not fit the extremal condition since two directed triangles can share an edge. We also notice that a directed cycle could still exist even if there is no bad event.

Applying the above framework, we get Algorithm 2 (which uses Algorithm 1) sampling partial acyclic orientations of G with probability proportional to $q^{\text{number of oriented edges}}$. An example of running Algorithm 2 is shown in Figure 4.4.

Algorithm 2 Sampling partial acyclic orientations

Assign a direction to each edge in G according to the distribution $[q, q, 1-2q]$.
while G contains a directed cycle **do**
    Assign a direction to each edge in G according to the distribution $[q, q, 1-2q]$.
    while triangle cycles exist **do**
        Let $\mathcal{R}$ be the set of triangle cycles
        Let $\mathcal{N}$ be the set of events that will not be resampled, be $\emptyset$.
        Let $\partial\mathcal{R}$ denote the boundary of $\mathcal{R}$ (neighbors of $\mathcal{R}$ outside $\mathcal{R}$) in the dependency graph.
        while $\partial\mathcal{R}\backslash\mathcal{N} \neq \emptyset$ **do**
            for each triangle $T \in \partial\mathcal{R}\backslash\mathcal{N}$ **do**
                if $\text{var}(T) \cap \text{var}(\mathcal{R}) = X_e$ and e is directed **then**
                    Add T to R
                else
                    Add T to $\mathcal{N}$.
    for each edge e in $\mathcal{R}$ **do**
        Resample e according to the distribution
Output G.

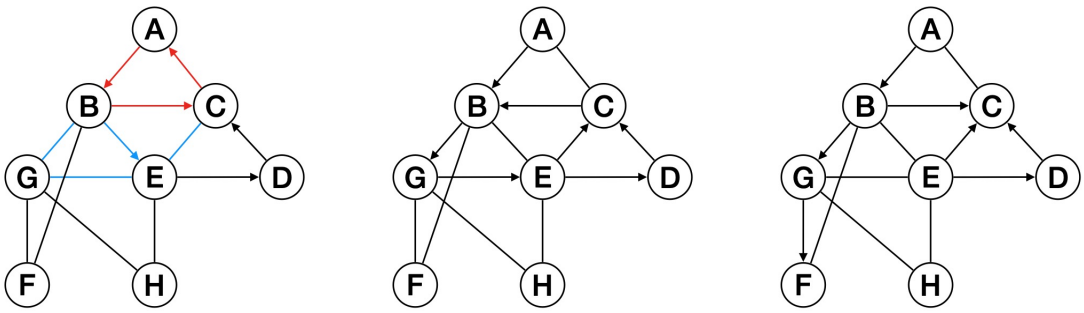

Figure 4.4: The orientation on the left has one bad event (the red directed triangle ABC). Blue triangles are on the boundary. Both red and blue edges need to be resampled. The middle orientation is an orientation after resampling. Even though there is no directed triangle, the graph is still acyclic (cycle BGEC). It means that we need to resample all edges. Finally, we get the right partial acyclic orientation.

Theorem 20. *Let $G = (V, E)$ be a chordal graph with maximum degree d. If $q \leq 0.24d^{-\frac{2}{3}}$, the expected number of variable resamples is $O(|E| \log |V|)$.*

Proof. We will apply Theorem 19. The probability that a specific bad event occurs is $p = 2q^3$, since the cycle can be oriented in two ways and it has three edges. Let A and B be two neighboring events. This can happen only if they share a single edge. Hence, for B to happen, the shared edge already has to be directed and its other two edges need to take the compatible direction. Thus, $r = q^2$. We claim that $\Delta \leq \frac{3}{2}d$. To see this, let A be a bad event with a maximum degree in the dependency graph. Let X_{e_1}, X_{e_2}, and X_{e_3} be the variables involved in A and let c_i be the number of other bad events that involve X_{e_i}. These other bad events correspond to different cycles of length three. If $e_1 = (v_1, v_2)$, $e_2 = (v_2, v_3)$, and $e_3 = (v_3, v_1)$, then v_2's degree is $c_1 + c_2 + 2$, where the $+2$ are for v_1 and v_3. Since we have $c_1 + c_2 + 2 \leq d$, $c_2 + c_3 + 2 \leq d$, $c_1 + c_3 + 2 \leq d$, and $c_1 + c_2 + c_3 = \Delta$, we get that $2(c + 1 + c_2 + c_3) + 6 = 2\Delta + 6 \leq 3d$, leading to $\Delta \leq \frac{3}{2}d$. Finally, from $6ep\Delta^2 \leq 1$ and $3er\Delta \leq 1$ we get $q \leq \left(3\sqrt[3]{e}d^{\frac{2}{3}}\right)^{-1} \leq 0.24d^{-\frac{2}{3}}$. Since the given chordal graph has $|E|$ edges and at most $\binom{|V|}{3}$ bad events, the expected number of variable resamples is $O(|E| \log |V|)$. □

The open problem is how many times do we need to resample all edges, in order to bound the overall running time. One of our original motivations for this work was to extend the partial rejection sampling framework to acyclic orientations. In a subsequent work, presented in Section 4.2, we found a more elegant and efficient algorithm for acyclic orientations in chordal graphs. While this particular motivation no longer applies to chordal graphs, we hope that the result presented here will be useful in other contexts as an illustration of the power and pitfalls of the partial rejection framework technique.

Chapter 5

Sampling Random Chordal Graphs

In this section, we present a Markov chain to sample chordal graphs uniformly at random. Inspired by the decomposition technique for bounding mixing times of Markov chains [48], we prove a polynomial mixing time bound for the Markov chain for chordal graphs with treewidth one and a fixed number of simplicial vertices.

5.1 Motivations and Related Works

The underlying structure of a Bayesian network, the most common graphical model for representing causal relationships among a set of variables, is a directed acyclic graph (DAG). In a DAG representation, a directed edge $X \to Y$ means that variable X is a cause of variable Y. Learning the underlying structure of a Bayesian network from data is a hot topic in research of probabilistic graphical models, with substantial recent progress [22, 55, 60, 83]. The main idea is to define a score for the Bayesian network, indicating how the network fits the given data, and then to try to find a Bayesian network with the highest score. For a given structure of the Bayesian network, there exists a class of structures that have the same score; this class is called the Markov equivalence class (MEC). All DAGs in a MEC share the same skeleton (that is, the underlying undirected graph) and v-structures, where a v-structure for variables (a, b, c) is an induced subgraph $a \to b \leftarrow c$ [76]. MEC can be represented by an essential graph, which contains both directed and undirected edges, an edge is directed if it has the same direction in all DAGs in the MEC, otherwise, it is undirected. The size of a MEC is the number of v-structure-free DAGs in it, and calculating the size of a MEC is crucial because it indicates how much additional information is needed in order to recover the

underlying structure. It was proved that calculating the size of a MEC can be achieved by counting the number of v-structure-free DAGs in undirected connected chordal graphs (UCCGs) [5, 29].

The existing approaches to find the size of a MEC include Markov chain Monte Carlo methods [7,28], exact counting by recursion [29, 30], and tree decomposition [25, 71, 83]. These algorithms are all tested on UCCGs that are generated by non-uniform samplers [49, 61, 62], which may result in a bias on their real performance. This motivates our study of sampling chordal graphs uniformly at random.

The closest #P-hardness results for chordal graphs are from Kijima et al. [41]. For chordal graphs $\underline{G}, \overline{G}$, Kijima et al. proved that counting chordal graphs in a chordal sandwhich $\Omega_{\mathrm{C}}(\bar{G}, \underline{G})$ is #P-complete, where $\Omega_C(\bar{G}, \underline{G}) \stackrel{\text{def.}}{=} \{G \mid G \text{ is chordal, } \underline{G} \subseteq G \subseteq \overline{G}, \text{ and } \subseteq \text{ denotes the subgraph relationship}\}$. The #P-completeness holds when $\underline{G}$ does not have any edges. Due to the computational hardness of counting, we consider sampling approaches for generating chordal graphs.

Since efficient algorithms exist for deciding whether a graph is chordal [59], a simple Markov chain can be used to sample chordal graphs with n vertices: let G be the current chordal graph, then:

- Choose an edge $e = (i, j)$ for $1 \leq i, j \leq n$ uniformly at random. Then there are three possible cases:
  - If $e \in G$ and $G - e$ is chordal, set $G' = G - e$.
  - If $e \notin G$ and $G + e$ is chordal, set $G' = G + e$.
  - Otherwise, set $G' = G$.
- Update $G = G'$ with probability $1/2$.

The Markov chain is symmetric, thus, if it is also ergodic, the underlying sampling distribution is uniform. Ergodicity follows from the fact that this Markov chain connects the state space (that is, the Markov chain is irreducible): One can use the theory of chordal graphs to find edges to remove, one by one, to get to a tree, and then design a mechanism to convert one tree into another tree. This procedure is reminiscent of what we describe below for chordal graphs of bounded treewidth, hence we focus on the bounded treewidth case in this work.

Kijima et al. [41] showed that the above Markov chain, when restricted to a sandwich scenario, is slow mixing for some pairs of chordal graphs $(\overline{G}, \underline{G})$. To the best of our knowledge, the mixing time is unknown when there are no sandwich restrictions (that is, when $\underline{G}$ is the graph of n isolated vertices and $\overline{G}$ is the complete graph).

5.2 Our contribution

From now on, we will assume that we have a constant r bounding the treewidth, and, slightly abusing the notation, we denote by Ω_n all connected chordal graphs with n vertices and a treewidth bound r. We aim to design a Markov chain that samples graphs from Ω_n uniformly at random.

Algorithm 3 One step transition of the Markov chain

```
Pick a vertex u uniformly at random (u.a.r.).
if u is simplicial then
    Pick a set W of r vertices other than u u.a.r.
    Choose a random nonempty subset W' of W.
    if W' forms a clique then
        Deattach u from its current neighbors.
        Attach u to W'.
```

Every step of this chain happens with the same probability, hence the Markov chain is symmetric. It can convert the current chordal graph into a tree by always choosing W' of size 1. Hence, to prove the connectivity of the chain, it remains to show how it can convert one tree into another tree. Therefore, it suffices to prove connectivity for $r = 1$.

Along the lines of the decomposition technique [48] for bounding mixing times, we will decompose our state space into sets Ω_n^k containing chordal graphs on n vertices with exactly k simplicial vertices (and treewidth bound $r = 1$).

We show that the above Markov chain, when restricted to only states in Ω_n^k (that is, any state not in Ω_n^k will be rejected), connects the state space and mixes rapidly, as long as $k \neq 2$ (for $k = 2$ every such chordal graph is a path) and $k \neq n - 1$ (star graphs).

We will use the following notation. For $G \in \Omega_n^k$, let S^G be the set of its simplicial vertices and let $S_0^G \subseteq S^G$ be the simplicial vertices whose neighbor is of degree 2. Whenever clear from the context, we will omit the superscript G.

Lemma 21. *The Markov chain over Ω_n^k for $3 \leq k \leq n - 2$ connects the state space, which has diameter $O(n^2)$.*

Proof. Let the vertex set be $\{v_1, \ldots, v_n\}$. We say that a graph in Ω_n^k is *canonical* if it has v_1 as the root, has edges (v_1, v_j) for $2 \leq j \leq k + 1$ and edges (v_{j+1}, v_j) for $k + 1 \leq j \leq n$.

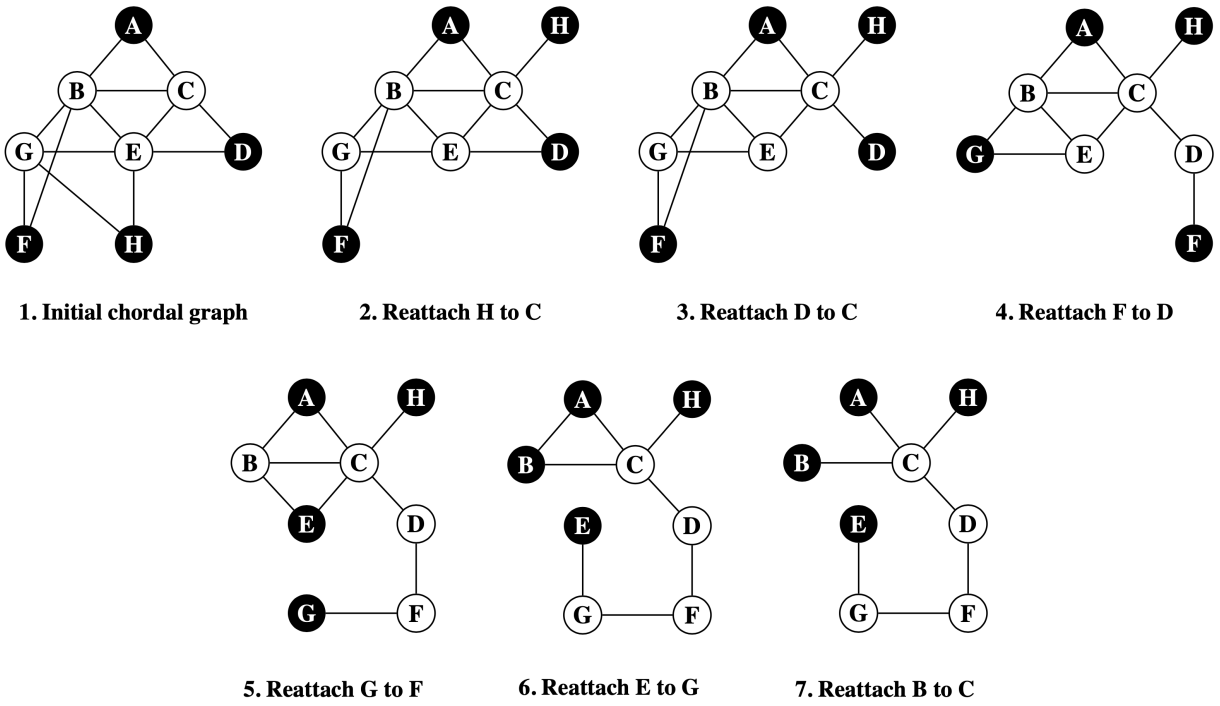

Figure 5.1: An example of converting a graph in Ω_8^4 to the "star with a tail" graph with vertex C as the star using the Markov chain transitions. Simplicial vertices are in black. After each step of the transitions, there are always four simplicial vertices.

We will first show that any graph in Ω_n^k can be transformed, using steps of the Markov chain, to a "star with a tail" graph with v_1 at the center of the star. The "star with a tail" graph consists of $k-1$ vertices attached directly to v_1 and a path of $n-k$ vertices also attached to v_1. Notice that the canonical graph is a special case of "star with a tail". An example is shown in Figure 5.1.

Let $G \in \Omega_n^k$. We will first achieve that $v_1 \notin S$. If $v_1 \in S$, and there is a $u \in S_0$, we just reconnect u to v_1 (notice that the number of simplicial vertices did not change, since u was in S_0). If there is no such u, then suppose we root the tree at v_1 and u be the leaf furthest from v_1. This leaf has siblings (since it is not in S_0), we reconnect each of the siblings to an arbitrary vertex in $G - S$. Now we have $u \in S_0$ and we proceed as before.

Let t be a simplicial vertex. We will process the current simplicial vertices one by one. For a simplicial vertex $u \neq t$, if $u \in S_0$, reconnect u to t and let $t := u$; otherwise reconnect u to v_1. Both changes will keep the number of simplicial vertices unchanged.

The final step is to transform the current "star with a tail" graph into the canonical form. While

there is a $u \leq k$ in the tail part, then there must exist a v in the star part such that $v > k$. If $u \in S$, we reconnect v to the parent of u, then reconnect u to v_1, keeping the number of simplicial vertices the same. Otherwise, when $u \notin S$, let $w \neq v$ be a vertex in the star (notice that it must exist since $k \geq 3$). Let $x_1, \ldots, x_p$ be the descendants of u, in this order (where $x_p \in S$). We remove them, one by one, and attach them to v, forming a path $x_p, \ldots, x_1$ of descendants of v. At this time u will be simplicial and we proceed with u as before and reconnect x_i's back to the tail.

Now vertices v_j for $k+1 \leq j \leq n$ are all in the tail, but may be in the wrong order. We first make v_{k+1} to be connected to v_1. We proceed through its descendants as before with a w_1 in the star, reconnect v_{k+1} to another w_2 in the star, proceed one by one with v_{k+1}'s ancestors to the path start at w_1. Then we reconnect v_{k+1} to v_1 and reconnect ancestors and descendants one by one back to v_{k+1}. We keep doing this until we get the correct order.

From the above, we know that at most $O(n)$ transitions are needed to move a vertex to its correct position. So the diameter of the state space is $O(n^2)$. □

Next we prove a mixing time bound within each subspace Ω_n^k.

Lemma 22. *The mixing time of the Markov chain on Ω_n^k for $3 \leq k \leq n-2$ is $O(kn \log n)$.*

Proof. We use the path coupling technique [15] to bound the mixing time. We will use moves of the Markov chain to induce the metric on the state space. In other words, two trees are at a distance that equal the smallest number of Markov chain moves that transform the first tree into the second tree.

Therefore, two trees X_t and Y_t are adjacent if they differ by exactly one Markov chain move. Let u be that different vertex, v_x be the parent of u in X_t, v_y be the parent of u in Y_t. The coupling chooses the same pair of vertices in both X_t and Y_t : the first vertex will be detached from its current neighbor and attached to the second vertex. There are two possible cases.

The first is $u \in S_0$, then the removal of u will make v_x, v_y simplicial. So, in this case the good moves (when the distance decreases) are X_t removes (u, v_x) , Y_t removes (u, v_y), and both add (u, w) and $w \in S$. There are $|S|$ good moves; bad moves are X_t and Y_t remove $w \in S_0$ and add (w, u). So the number of bad moves is $|S_0| - 1$. Obviously $|S_0| - 1 < |S|$.

Another possibility is $u \in S_{\geq 1}$, in which case good moves are X_t removes (u, v_x) , Y_t removes (u, v_y), and both add (u, w) and $w \notin S$. So there are $n - |S| + 1$ good moves. And the number of bad

moves is still $|S_0| - 1$. We will show by induction that $n - |S| + 1 > |S_0| - 1$, which is equivalent to showing

$$n > |S_0| + |S| - 2 = 2\,|S_0| + |S_{\geq 1}| - 2 \tag{5.1}$$

1. If $|S_{\geq 1}| = 0$, then for each $v \in S_0$ its parent is nonsimplicial, so $n \geq 2\,|S_0| + 1$.
2. If $|S_0| = 0$, (5.1) is obviously true.
3. Assume (5.1) holds for step $t \geq 0$ and at step $t+1$ a new vertex is connected to a vertex v in the tree. If $v \in S$ then $S \mid$ will not change and $|S_0|$ will increase by one if and only if $v \in S_{\geq 1}$; if $v \notin S$, then $|S_0|$ will not change and $|S_{\geq 1}|$ will increase by one. Therefore, (5.1) holds at step $t+1$ in both cases.

Therefore, in every case, the number of good moves is strictly larger than the number of bad moves. Plugging this in [15] yields a mixing time of $O(kn \log n)$. □

We showed that the Markov chain, when restricted to trees with k leaves (these are exactly the simplicial vertices), mixes rapidly. A natural open question is whether the Markov chain mixes rapidly for all (labeled) trees, though it should be noted that there is an exact formula (the Caley's formula) for the number of all labeled trees with n vertices, and therefore we find the restricted version of the Markov chain more exciting to study. We conclude the chapter with a generalization of the above open problem: Is the mixing time of the Markov chain polynomial for an arbitrary (constant) treewidth bound r? And, what is the mixing time of the edge add/remove chain?

Chapter 6

Conclusions

Counting and sampling of different types of graph structures on general graphs have been widely studied. Most of the studied graph structures are hard (unlikely to be exactly solved or approximately solved) on general graphs as well as restricted graphs like bounded-degree graphs and planar graphs. Finding underlying graphs that make counting and sampling of those graph structures polynomial-time solvable helps us understand both the graph and the structures better.

In Chapter 3 we studied the classical problem of sampling independent sets. We showed that the well-known Dyer&Greenhill chain can sample weighted independent sets in polynomial time on chordal graphs with bounded separator sizes. Also, the bound of mixing time we got gives us a strong hint that the Markov chain might mix in polynomial time on chordal graphs without a separator bound. We leave the problem of bounding the mixing time on general chordal graphs open.

In Chapter 4 we first presented a polynomial-time counting framework for acyclic, bipolar, sink-free, source-sink-free orientations on chordal graphs through a manner of applying dynamic programming over a clique tree. For acyclic and bipolar orientations, we also designed polynomial-time uniform samplers for them based on the counter we developed. We also introduced source-sink-free orientations, which are a natural restricted version of sink-free orientations. Strong orientations are a special case of source-sink-free orientations since they have no sources and no sinks. Unlike acyclic and bipolar orientations, we cannot convert the counter to a uniform sampler because of a two-level inclusion-exclusion approach used for the counter. We leave uniform sampling of source-sink-free orientations as an open problem. Besides this, we are also interested in designing a sampler and a counter for strong orientations. In the remainder of the chapter, we presented a weighted sampler

for so-called "partial acyclic orientations" on chordal graphs using the partial rejection sampling framework.

In Chapter 5, we designed a Markov chain to sample random chordal graphs with fixed number of vertices. We proved a polynomial mixing time bound for chordal graphs with treewidth 1 and fixed number of simplicial vertices. A natural open problem is to bound the mixing time of the chain for general chordal graphs.

Bibliography

[1] Geir Agnarsson. On chordal graphs and their chromatic polynomials. *Mathematica Scandinavica*, pages 240–246, 2003.

[2] David Aldous. Random walks on finite groups and rapidly mixing Markov chains. In *Séminaire de Probabilités XVII 1981/82*, pages 243–297. Springer, 1983.

[3] Noga Alon, Alan Frieze, and Dominic Welsh. Polynomial time randomized approximation schemes for Tutte–gröthendieck invariants: the dense case. *Random Structures & Algorithms*, 6(4):459–478, 1995.

[4] Nima Anari, Kuikui Liu, and Shayan Oveis Gharan. Spectral independence in high-dimensional expanders and applications to the hardcore model. In *61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1319–1330. IEEE, 2020.

[5] Steen A Andersson, David Madigan, and Michael D Perlman. A characterization of Markov equivalence classes for acyclic digraphs. *The Annals of Statistics*, 25(2):505–541, 1997.

[6] VS Anil Kumar and Hariharan Ramesh. Coupling vs. conductance for the jerrum–sinclair chain. *Random Structures & Algorithms*, 18(1):1–17, 2001.

[7] Megan Bernstein and Prasad Tetali. On sampling graphical Markov models. *arXiv preprint arXiv:1705.09717*, 2017.

[8] Ivona Bezáková and Wenbo Sun. Mixing of Markov chains for independent sets on chordal graphs with bounded separators. In *International Computing and Combinatorics Conference (COCOON)*, pages 664–676. Springer, 2020.

[9] Ivona Bezáková and Wenbo Sun. Counting and sampling orientations on chordal graphs. In *International Conference and Workshops on Algorithms and Computation (WALCOM)*, pages 352–364. Springer, 2022.

[10] Jean RS Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, pages 1–29. Springer, 1993.

[11] Magnus Bordewich and Ross J Kang. Rapid mixing of subset Glauber dynamics on graphs of bounded tree-width. In *International Colloquium on Automata, Languages, and Programming*, pages 533–544. Springer, 2011.

[12] Magnus Bordewich and Ross J. Kang. Subset Glauber dynamics on graphs, hypergraphs and matroids of bounded tree-width. *Electr. J. Comb.*, 21(4):P4.19, 2014. Earlier version in ICALP '11.

[13] Peter B Borwein. On the complexity of calculating factorials. *Journal of Algorithms*, 6(3):376–380, 1985.

[14] Thomas Brylawski and James Oxley. The Tutte polynomial and its applications. *Matroid applications*, 40:123–225, 1992.

[15] Russ Bubley and Martin Dyer. Graph orientations with no sink and an approximation for a hard case of #SAT. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 248–257, 1997.

[16] Zongchen Chen, Kuikui Liu, and Eric Vigoda. Optimal mixing of glauber dynamics: Entropy factorization via high-dimensional expansion. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1537–1550, 2021.

[17] Henry Cohn, Robin Pemantle, and James Propp. Generating a random sink-free orientation in quadratic time. *arXiv preprint math/0103189*, 2001.

[18] Martin Dyer, Leslie Ann Goldberg, Catherine Greenhill, and Mark Jerrum. On the relative complexity of approximate counting problems. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 108–119. Springer, 2000.

[19] Martin Dyer and Catherine Greenhill. On Markov chains for independent sets. *Journal of Algorithms*, 35(1):17–49, 2000.

[20] Martin Dyer, Catherine Greenhill, and Haiko Müller. Counting independent sets in graphs with bounded bipartite pathwidth. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 298–310. Springer, 2019.

[21] Martin E. Dyer, Alan M. Frieze, and Mark Jerrum. On counting independent sets in sparse graphs. *SIAM J. Comput.*, 31(5):1527–1541, 2002.

[22] Nir Friedman and Daphne Koller. Being bayesian about network structure. a bayesian approach to structure discovery in bayesian networks. *Machine learning*, 50(1):95–125, 2003.

[23] Andreas Galanis, Qi Ge, Daniel Štefankovič, Eric Vigoda, and Linji Yang. Improved inapproximability results for counting independent sets in the hard-core model. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 567–578. Springer, 2011.

[24] Qi Ge and Daniel Štefankovič. A graph polynomial for independent sets of bipartite graphs. *Comb. Probab. Comput.*, 21(5):695–714, 2012.

[25] AmirEmad Ghassami, Saber Salehkaleybar, Negar Kiyavash, and Kun Zhang. Counting and sampling from Markov equivalent dags using clique trees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3664–3671, 2019.

[26] Leslie Ann Goldberg and Mark Jerrum. Inapproximability of the Tutte polynomial. *Information and Computation*, 206(7):908–929, 2008.

[27] Heng Guo, Mark Jerrum, and Jingcheng Liu. Uniform sampling through the Lovász local lemma. *Journal of the ACM (JACM)*, 66(3):1–31, 2019.

[28] Yangbo He, Jinzhu Jia, and Bin Yu. Reversible mcmc on Markov equivalence classes of sparse directed acyclic graphs. *The Annals of Statistics*, 41(4):1742–1779, 2013.

[29] Yangbo He, Jinzhu Jia, and Bin Yu. Counting and exploring sizes of Markov equivalence classes of directed acyclic graphs. *The Journal of Machine Learning Research*, 16(1):2589–2609, 2015.

[30] Yangbo He and Bin Yu. Formulas for counting the sizes of Markov equivalence classes of directed acyclic graphs. *arXiv preprint arXiv:1610.07921*, 2016.

[31] Mark Huber. Exact sampling and approximate counting techniques. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 31–40, 1998.

[32] François Jaeger, Dirk Vertigan, and Dominic Welsh. On the computational complexity of the jones and Tutte polynomials. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 108, pages 35–53. Cambridge University Press, 1990.

[33] Mark Jerrum. *Counting, Sampling and Integrating: Algorithms and Complexity*. Birhäuser, 2003.

[34] Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18(6):1149–1178, 1989.

[35] Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the ising model. *SIAM Journal on computing*, 22(5):1087–1116, 1993.

[36] Mark Jerrum and Alistair Sinclair. The Markov chain Monte Carlo method: an approach to approximate counting and integration. *Approximation Algorithms for NP-hard problems, PWS Publishing*, 1996.

[37] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM (JACM)*, 51(4):671–697, 2004.

[38] Mark Jerrum, Leslie G Valiant, and Vijay Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

[39] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[40] Pieter Kasteleyn. Graph theory and crystal physics. *Graph theory and theoretical physics*, pages 43–110, 1967.

[41] Shuji Kijima, Masashi Kiyomi, Yoshio Okamoto, and Takeaki Uno. On listing, sampling, and counting the chordal graphs with edge constraints. In *International Computing and Combinatorics Conference*, pages 458–467. Springer, 2008.

[42] Gustav Kirchhoff. Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148(12):497–508, 1847.

[43] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[44] Michel Las Vergnas. Convexity in oriented matroids. *Journal of Combinatorial Theory, Series B*, 29(2):231–243, 1980.

[45] Abraham Lempel. An algorithm for planarity testing of graphs. *Theory of Graphs, International Syposium, Rome, July 1966, Rosenstiel, P. edit.*, 1967.

[46] Nathan Linial. Hard enumeration problems in geometry and combinatorics. *SIAM Journal on Algebraic Discrete Methods*, 7(2):331–335, 1986.

[47] Michael Luby and Eric Vigoda. Fast convergence of the Glauber dynamics for sampling independent sets. *Random Structures & Algorithms*, 15(3-4):229–241, 1999.

[48] Neal Madras and Dana Randall. Markov chain decomposition for convergence rate analysis. *Annals of Applied Probability*, pages 581–606, 2002.

[49] Lilian Markenzon, Oswaldo Vernet, and Luiz Henrique Araujo. Two methods for the generation of chordal graphs. *Annals of Operations Research*, 157(1):47–60, 2008.

[50] J Matthews. *Markov chains for sampling matchings*. PhD Thesis, University of Edinburgh, 2008.

[51] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[52] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[53] Robin A Moser and Gábor Tardos. A constructive proof of the general lovász local lemma. *Journal of the ACM (JACM)*, 57(2):1–15, 2010.

[54] Joseph Naor, Moni Naor, and Alejandro A Schäffer. Fast parallel algorithms for chordal graphs. *SIAM Journal on Computing*, 18(2):327–349, 1989.

[55] Siqi Nie, Denis D Mauá, Cassio P De Campos, and Qiang Ji. Advances in learning bayesian networks of bounded treewidth. *Advances in neural information processing systems*, 27, 2014.

[56] Yoshio Okamoto, Takeaki Uno, and Ryuhei Uehara. Linear-time counting algorithms for independent sets in chordal graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 433–444. Springer, 2005.

[57] Yoshio Okamoto, Takeaki Uno, and Ryuhei Uehara. Counting the number of independent sets in chordal graphs. *Journal of Discrete Algorithms*, 6(2):229–242, 2008.

[58] James Gary Propp and David Bruce Wilson. How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph. *Journal of Algorithms*, 27(2):170–217, 1998.

[59] Donald J Rose, R Endre Tarjan, and George S Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing*, 5(2):266–283, 1976.

[60] Mauro Scanagatta, Cassio P de Campos, Giorgio Corani, and Marco Zaffalon. Learning bayesian networks with thousands of variables. *Advances in neural information processing systems*, 28, 2015.

[61] Oylum Şeker, Pinar Heggernes, Tınaz Ekim, and Z Caner Taşkın. Linear-time generation of random chordal graphs. In *International Conference on Algorithms and Complexity*, pages 442–453. Springer, 2017.

[62] Oylum Şeker, Pinar Heggernes, Tınaz Ekim, and Z Caner Taşkın. Generation of random chordal graphs using subtrees of a tree. *arXiv preprint arXiv:1810.13326*, 2018.

[63] Alistair Sinclair. Improved bounds for mixing rates of Markov chains and multicommodity flow. *Combinatorics, probability and Computing*, 1(4):351–370, 1992.

[64] Alistair Sinclair, Piyush Srivastava, Daniel Štefankovič, and Yitong Yin. Spatial mixing and the connective constant: Optimal bounds. *Probability Theory and Related Fields*, 168(1):153–197, 2017.

[65] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[66] Allan Sly. Computational transition at the uniqueness threshold. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 287–296. IEEE, 2010.

[67] Richard P Stanley. Acyclic orientations of graphs. *Discrete Mathematics*, 5(2):171–178, 1973.

[68] Wenbo Sun. Sampling and counting acyclic orientations in chordal graphs (student abstract). In *Thirty-Sixth AAAI Conference on Artificial Intelligence*, 2022.

[69] Wenbo Sun and Ivona Bezáková. Sampling random chordal graphs by MCMC (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13929–13930, 2020.

[70] Wenbo Sun and Ivona Bezáková. Sampling partial acyclic orientations in chordal graphs by the lovasz local lemma (student abstract). In *Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2021.

[71] Topi Talvitie and Mikko Koivisto. Counting and sampling Markov equivalent directed acyclic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7984–7991, 2019.

[72] Robert Endre Tarjan. Two streamlined depth-first search algorithms. *Fundamenta Informaticae*, 9(1):85–94, 1986.

[73] WT Tutte. Graph-polynomials. *Advances in Applied Mathematics*, 32(1-2):5–9, 2004.

[74] Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[75] Lieven Vandenberghe, Martin S Andersen, et al. Chordal graphs and semidefinite optimization. *Foundations and Trends in Optimization*, 1(4):241–433, 2015.

[76] Thomas Verma and Judea Pearl. Equivalence and synthesis of causal models. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, pages 255–270, 1990.

[77] Dirk Vertigan. The computational complexity of Tutte invariants for planar graphs. *SIAM Journal on Computing*, 35(3):690–712, 2005.

[78] Dirk Vertigan and Dominic Welsh. The computational complexity of the Tutte plane: the bipartite case. *Combinatorics, Probability and Computing*, 1(2):181–187, 1992.

[79] Eric Vigoda. A note on the Glauber dynamics for sampling independent sets. *Electr. J. Comb.*, 8(1), 2001.

[80] Dror Weitz. Counting independent sets up to the tree threshold. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 140–149, 2006.

[81] Dominic Welsh. Approximate counting. *Surveys in Combinatorics*, 241:287–324, 1997.

[82] Dominic Welsh. The Tutte polynomial. *Random Structures & Algorithms*, 15(3-4):210–228, 1999.

[83] Marcel Wienöbst, Max Bannach, and Maciej Liskiewicz. Polynomial-time algorithms for counting and sampling Markov equivalent dags. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12198–12206, 2021.