Counting and Sampling Orientations on Chordal Graphs

Ivona Bezáková* and Wenbo Sun

Rochester Institute of Technology, Rochester NY 14623, USA

Abstract. We study counting problems for several types of orientations of chordal graphs: source-sink-free orientations, sink-free orientations, acyclic orientations, and bipolar orientations, and, for the latter two, we also present linear-time uniform samplers. Counting sink-free, acyclic, or bipolar orientations are known to be #P-complete for general graphs, motivating our study on a restricted, yet well-studied, graph class. Our main focus is source-sink-free orientations, a natural restricted version of sink-free orientations related to strong orientations, which we introduce in this work. These orientations are intriguing, since despite their similarity, currently known FPRAS and sampling techniques (such as Markov chains or sink-popping) that apply to sink-free orientations do not seem to apply to source-sink-free orientations. We present fast polynomial-time algorithms counting these orientations on chordal graphs. Our approach combines dynamic programming with inclusion-exclusion (going two levels deep for source-sink-free orientations and one level for sink-free orientations) throughout the computation. Dynamic programming counting algorithms can be typically used to produce a uniformly random sample. However, due to the negative terms of the inclusion-exclusion, the typical approach to obtain a polynomial-time sampling algorithm does not apply in our case. Obtaining such an almost uniform sampling algorithm for source-sink-free orientations in chordal graphs remains an open problem.

Little is known about counting or sampling of acyclic or bipolar orientations, even on restricted graph classes. We design efficient (linear-time) exact uniform sampling algorithms for these orientations on chordal graphs. These algorithms are a byproduct of our counting algorithms, but unlike in other works that provide dynamic-programming-based samplers, we produce a random orientation without computing the corresponding count, which leads to a faster running time than the counting algorithm (since it avoids manipulation of large integers).

1 Introduction

An orientation of an undirected graph is an assignment of a direction to each edge, converting the original graph to a directed graph. We initiate the study of counting *source-sink-free* orientations, where there are no sources, nor sinks (that

* Partially supported by NSF awards 1819546 and 1821459.

is, no vertices of indegree or outdegree 0). Our motivation is twofold: First, these orientations are related to the well-studied *sink-free* orientations, which can be counted (approximately) despite being #P-hard for general graphs. While clearly similar, source-sink-free orientations exhibit certain properties that prevent the application of the current techniques for counting or sampling of sink-free orientations. Therefore, new techniques are needed to understand this problem, and we are starting with a restricted but well-known graph class. Second, source-sink-free orientations can be thought of as a local (soft) version of strong orientations, another well-studied class of orientations, the counting of which is also #P-hard on general graphs [15] as well as on restricted graph classes such as planar and bipartite graphs [13]. Our study is a first step beyond sink-free and towards strong orientations.

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 [9, 16]), on which some problems that are NP-hard or #P-hard on general graphs can be solved in polynomial time; see, for example, [11].

Sink-free orientations are well understood. Bubley and Dyer [5] proved that counting these orientations is #P-complete on general graphs. They also provided 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 [8] used the "coupling from the past" technique to obtain an exact sample in time $O(|E(G)|^4)$. Cohn, Pemantle, and Propp [6] 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 [7], yielding a uniformly random sink-free orientation in time $O(|V(G)|^2)$ time. Interestingly, none of these techniques appear to apply to the problems of counting and sampling of source-sink-free orientations.

Our main contribution is a polynomial (cubic in the worst case) exact counting algorithm for source-sink-free orientations in chordal graphs, combining dynamic programming with two-level inclusion-exclusion at every step of the dynamic programming computation. However, our combination with inclusion-exclusion prevents us from extending our algorithm to an exact uniform sampler, and we leave the sampling question as the main open problem of our work. We apply a similar approach, using one-level inclusion-exclusion, to count sink-free orientations of a given chordal graph in almost linear time, significantly improving the running time over the FPRAS for this graph class. Besides these two orientations, we also present almost linear time counting and linear time sampling algorithms for acyclic orientations and bipolar orientations, which are both

#P-complete on general graphs [10]. The problem of counting acyclic orientations has also attracted a lot of attention since it corresponds to a specific input of the well-studied Tutte polynomial $T_G(x, y)$ [4], in particular to $T_G(2, 0)$ [14]), which plays an important role in graph theory and statistical physics. Acyclic orientations can be counted efficiently on chordal graphs via the calculation of the chromatic polynomial [1]. However, this result does not yield a(n almost) uniform sampler since acyclic orientations are not known to be self-reducible. In this work we present a simple linear-time exact uniform sampling algorithm for acyclic orientations in chordal graphs. An interesting aspect of this sampling algorithm is that it runs faster than its counting counterpart. This is atypical — dynamic programming based samplers usually rely on a precomputation of the corresponding counts and are efficient only after substantial preprocessing time. We note that, with some extra work to maintain the desired (unique) source s and sink t, our results extend to counting and sampling of bipolar (s, t)-orientations, also known as st-numberings. Finally, we compare our work to the recent celebrated results of Wienöbst, Bannach, and Liśkiewicz [16], who count and sample another type of orientations, the so-called v-structure-free acyclic (or moral acyclic) orientations in chordal graphs. The authors prove interesting structural results for these orientations and employ dynamic programming over the clique tree in order to count them. In their case, the dynamic programming consists of additive quantities, which allows them to extend their counting approach to sampling. In contrast, in our work we either do not need to compute the counts in order to sample, or our dynamic programming does not appear to extend to sampling due to the presence of negative terms in the computation.

The paper is organized as follows. Section 2 contains preliminaries on chordal graphs and clique trees. Our main result, a fast counting algorithm for source-sink-free orientations, combining dynamic programming with inclusion-exclusion, is in Section 3. We summarize our other results in Section 4.

2 Preliminaries

For a graph G, we denote by $G[U]$ the graph induced in G on the vertex set $U \subseteq V(G)$. 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. 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, see, for example, [12]. 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* $\mathrm{Sep}(C) = C \cap \mathrm{Parent}(C)$ and a *residual set* $\mathrm{Res}(C) = C \setminus \mathrm{Sep}(C)$, where $\mathrm{Parent}(C)$ 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, [2, 12].

- 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}, excluding the vertices in $\text{Sep}(C)$, i.e., $D(C) := C' \in V(T_C)C' - \text{Sep}(C)$. Let $A(C)$ be the vertices in the cliques not in T_C, excluding the vertices in $\text{Sep}(C)$, i.e., $A(C) := \bigcup_{C' \in V(T_{C_r}) - V(T_C)} C' - \text{Sep}(C)$. The separator $\text{Sep}(C)$ 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 $O(|E(G)|)$.

We use $G[T_C]$ for the subgraph induced by the vertices that belong to cliques in T_C, i.e., $G[T_C] := G[\bigcup_{C' \in V(T_C)} C']$. 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 $\text{Sep}(C)$ removed, i.e., $\hat{G}[T_C] := G[T_C] - E(G[\text{Sep}(C)])$.

The following lemma will be essential for our calculations.

Lemma 1. *Let C be a clique in the rooted clique tree T_{C_r} and let $C_1, C_2, \ldots, C_d$ be its child 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 $\{1, 2, \ldots, d\}$.

3 Counting source-sink-free orientations

In this section we describe the main contribution of this paper. We show how to count source-sink-free orientations in chordal graphs using dynamic programming on the clique tree. While this approach is quite standard for algorithms on

chordal graphs, the novel aspect of our work is to employ a two-level inclusion-exclusion principle as a subroutine of the dynamic programming. We prove the following theorem:

Theorem 1. *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.*

Proof. We define the following quantities for each clique C in a rooted clique tree T of the given chordal graph G:

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

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_{C_r}. The quantity $\mathrm{SSFO}(T_{C_r})$ represents the number of source-sink-free orientations of G.

To simplify our expressions, for a clique C we define quantities $\mathrm{oa}(C)$, $\mathrm{os}(C)$, $\mathrm{ox}(C)$, $\mathrm{oss}(C)$, and $\mathrm{oxx}(C)$ as follows. Let $\mathrm{oa}(C)$ be the number of all orientations of C, i.e., $\mathrm{oa}(C) = 2^{\binom{|C|}{2}}$. Let $\mathrm{os}(C, v)$ be the number of orientations of 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 simplify the notation to just $\mathrm{os}(C)$. 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, we get that $\mathrm{ox}(C) = \mathrm{oa}(C) - |C|\,\mathrm{os}(C)$. 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 $\mathrm{oss}(C)$. 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, $\mathrm{oxx}(C) = \mathrm{oa}(C) - 2|C|\,\mathrm{os}(C) + \binom{|C|}{2}\,\mathrm{oss}(C)$.

Base case of the computation of $\mathrm{SSFO}(T_C)$, $\mathrm{SoO}(T_C, v_1)$, $\mathrm{SiO}(T_C, v_2)$, and $\mathrm{SoSiO}(T_C, v_1, v_2)$: 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 four mutually exclusive cases.

▶ The orientation restricted to $G[\mathrm{Res}(C)]$ contains no sources or 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.

▶ 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.

▶ 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.

▶ 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 child cliques in T. For $u \in \mathrm{Res}(C)$ we denote by I_u the set of indices corresponding to the child cliques containing u, i.e., $I_u := \{i \in [d] \mid u \in C_i\}$.

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, or a 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:

▶ There is no sink in the orientation restricted to $G[\mathrm{Res}(C)]$. There are $\alpha_1 := \mathrm{ox}(\mathrm{Res}(C))2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}\prod_{i=1}^{d}\mathrm{SSFO}(T_{C_i})$ such orientations of $\hat{G}[T_C]$.

▶ 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 child subtrees. The number of orientations of $\hat{G}[T_C]$ corresponding to this case is

$$\alpha_2(u_2) := \mathrm{os}(\mathrm{Res}(C), u_2)(2^{|\,\mathrm{Sep}(C)|}-1)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}\prod_{i=1}^{d}\mathrm{SSFO}(T_{C_i}) + \mathrm{os}(\mathrm{Res}(C), u_2)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}\prod_{i\in[d]-I_{u_2}}\mathrm{SSFO}(T_{C_i})\times\left(\prod_{i\in I_{u_2}}\mathrm{SSFO}(T_{C_i}) - \prod_{i\in I_{u_2}}\mathrm{SiO}(T_{C_i}, u_2)\right).$$

Putting the two cases together, we get $\mathrm{SoO}(T_C, v_1) = \alpha_1 + \sum_{u_2\in\mathrm{Res}(C)}\alpha_2(u_2)$. Note that $\mathrm{SiO}(T_C, v_1)$ can be computed analogously.

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

▶ The orientation restricted to $G[\mathrm{Res}(C)]$ contains no sources or sinks. Then, all the remaining edges within $\mathrm{Res}(C)$ can be oriented arbitrarily, and the child 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 $\beta_1 := \mathrm{ox}(\mathrm{Res}(C))2^{|\,\mathrm{Sep}(C)||\,\mathrm{Res}(C)|}\prod_{i=1}^{d}\mathrm{SSFO}(T_{C_i})$.

▶ 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 child 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:

$$\beta_2(u_1) := \mathrm{ox}(\mathrm{Res}(C)^{-1})(2^{|\,\mathrm{Sep}(C)|}-1)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}\prod_{i=1}^{d}\mathrm{SSFO}(T_{C_i}) + \mathrm{ox}(\mathrm{Res}(C)^{-1})2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)}\prod_{i\in[d]-I_{u_1}}\mathrm{SSFO}(T_{C_i})\times\left(\prod_{i\in I_{u_1}}\mathrm{SSFO}(T_{C_i}) - \prod_{i\in I_{u_1}}\mathrm{SoO}(T_{C_i}, u_1)\right).$$

▶ The orientation restricted to $G[\mathrm{Res}(C)]$ contains a (single) sink u_2 and no sources. The number of the corresponding orientations of $\hat{G}[T_C]$ can be computed analogously to the previous case; we refer to this quantity as $\beta_3(u_2)$.

▶ 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

$$\beta_{4a} := \mathrm{oss}(\mathrm{Res}(C))(2^{|\,\mathrm{Sep}(C)|}-1)^2 2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-2)} \prod_{i=1}^{d} \mathrm{SSFO}(T_{C_i}).$$

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 child subtrees. The number of corresponding orientations is

$$\beta_{4b}(u_2) := \mathrm{oss}(\mathrm{Res}(C))(2^{|\,\mathrm{Sep}(C)|}-1)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-2)} \times \prod_{i\in[d]-I_{u_2}} \mathrm{SSFO}(T_{C_i}) \left(\prod_{i\in I_{u_2}} \mathrm{SSFO}(T_{C_i}) - \prod_{i\in I_{u_2}} \mathrm{SiO}(T_{C_i}, u_2) \right).$$

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). The number of corresponding orientations can be computed analogously to the previous subcase; we refer to this quantity as $\beta_{4c}(u_1)$.

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 child 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 or 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 d, 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 the double subtraction. Therefore, the number of orientations of $\hat{G}[T_C]$ corresponding to case d is

$$\beta_{4d}(u_1,u_2) := \mathrm{oss}(\mathrm{Res}(C))2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-2)} \times \left(\prod_{i\in[d]} \mathrm{SSFO}(T_{C_i}) - |X_{u_1}| - |Y_{u_2}| + |Z_{u_1,u_2}|\right).$$

Putting it all together we get

$$\mathrm{SSFO}(T_C) = \beta_1 + \sum_{u_1\in\mathrm{Res}(C)} \beta_2(u_1) + \sum_{u_2\in\mathrm{Res}(C)} \beta_3(u_2) + \sum_{u_1,u_2\in\mathrm{Res}(C),u_1\neq u_2} [\beta_{4a} + \beta_{4b}(u_2) + \beta_{4c}(u_1) + \beta_{4d}(u_1,u_2)].$$

Finally, we need to estimate the running time of the algorithm. After constructing the clique tree T (which is of size $O(|V(G)|)$), the algorithm performs a tree traversal of T. In the base case it performs $O(1)$ arithmetic operations per leaf clique.[1] To analyze the running time in the inductive case, we first pretend that we have access to the quantities $\mathrm{SSFO}(T_{C_i})$. Notice that we do not need to store the quantities $\mathrm{SoSiO}(T_C, v_1, v_2)$ for each v_1, v_2, since the computation is independent of v_1, v_2 and therefore we really need only one quantity $\mathrm{SoSiO}(T_C)$ for each clique. The computation of $\mathrm{SoSiO}(T_C)$ takes $O(d)$ arithmetic operations, assuming $\mathrm{SSFO}(T_{C_i})$'s have been computed. Since $d = \mathrm{outdeg}_T(C)$, the computation of all SoSiO's across the entire tree T takes $O(\sum_{C\in T} \mathrm{outdeg}_T(C)) = O(|T|) = O(|V(G)|)$ arithmetic operations.

Next we consider the computations of $\mathrm{SoO}(T_C, v_1)$; notice that the computations are independent of v_1. All α_1 quantities can be computed in $O(|V(G)|$ time following the same reasoning as for SoSiO. The same holds for the first term of the quantities $\alpha_2(u_2)$. We need to estimate the running time needed to compute the second (additive) term of the $\alpha_2(u_2)$'s. Notice that the size of the set I_{u_2} corresponding to the child cliques of C that contain u_2 is upper-bounded by $\deg_G(u_2)$. This is because for each child clique C_i for $i \in I_{u_2}$ we have a $w_i \in \mathrm{Res}(C_i)$. All the w_i's are distinct since $\mathrm{Sep}(C)$ separates them, yielding $|I_{u_2}| \leq \deg_G(u_2)$. We can rewrite the computation of the second term of $\alpha_2(u_2)$ as a product of $\mathrm{os}(\mathrm{Res}(C), u_2)2^{|\,\mathrm{Sep}(C)|(|\,\mathrm{Res}(C)|-1)} \prod_{i\in[d]} \mathrm{SSFO}(T_{C_i})$ and $(1 - \prod_{i\in I_{u_2}} \frac{\mathrm{SiO}(T_{C_i},u_2)}{\mathrm{SSFO}(T_{C_i}}))$. Computing the last term of this product 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 $O(\sum_{C\in T} \sum_{u_2\in\mathrm{Res}(C)} \deg_G(u_2)) = O(\sum_{u_2\in V(G)} \deg_G(u_2)) = O(|E(G)|)$ operations to compute all $\mathrm{SoO}(T_C)$'s (and the same holds for the $SiO(T_C)$'s).

[1] Computation of the factorial of a k-bit number takes $O(k \,\mathrm{polylog}\, k)$, see [3], which will be subsumed by our $\tilde{O}()$ notation.

It remains to bound the running time needed to compute all the SSFO's. Using the same arguments as before we get that the computation of all β_1, $\beta_2(u_1)$, $\beta_3(u_2)$, β_{4a}, $\beta_{4b}(u_2)$, and $\beta_{4c}(u_1)$ takes $O(|E(G)|)$ arithmetic operations. The tricky part is to account for the computation of the $\beta_{4d}(u_1, u_2)$'s, due to the 2-level inclusion-exclusion depth. In particular, we want to efficiently compute

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

$$\prod_{i\in[d]-I_{u_2}} \mathrm{SSFO}(T_{C_i}) \prod_{i\in I_{u_2}} \mathrm{SiO}(T_{C_i}, u_2) + \prod_{i\in[d]-I_{u_1}-I_{u_2}} \mathrm{SSFO}(T_{C_i}) \times$$

$$\prod_{i\in I_{u_1}-I_{u_2}} \mathrm{SoO}(T_{C_i}, u_1) \prod_{i\in I_{u_2}-I_{u_1}} \mathrm{SiO}(T_{C_i}, u_2) \prod_{i\in I_{u_1}\cap I_{u_2}} \mathrm{SoSiO}(T_{C_i}, u_1, u_2)] =$$

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

$$\prod_{i\in I_{u_1}-I_{u_2}} \frac{\mathrm{SoO}(T_{C_i}, u_1)}{\mathrm{SSFO}(T_{C_i})} \prod_{i\in I_{u_2}-I_{u_1}} \frac{\mathrm{SiO}(T_{C_i}, u_2)}{\mathrm{SSFO}(T_{C_i})} \prod_{i\in I_{u_1}\cap I_{u_2}} \frac{\mathrm{SoSiO}(T_{C_i}, u_1, u_2)}{\mathrm{SSFO}(T_{C_i})}].$$

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 this bound on the number of arithmetic operations across all cliques in T:

$$O(\sum_{C\in T} \sum_{u_1,u_2\in \mathrm{Res}(C)} [\deg_G(u_1) + \deg_G(u_2)]) = O(\sum_{C\in T} \sum_{u_1\in \mathrm{Res}(C)} |C| \deg_G(u_1)),$$

which is $O(|C_{\max}||E(G)|)$. This concludes the proof of the theorem. □

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 Results for the other types of orientations

In this section we briefly sketch our results on counting and sampling of acyclic and bipolar orientations, and on counting sink-free orientations. We include the detailed proofs and discussion in the appendix.

An orientation is *acyclic* if it does not contain a directed cycle. A structural examination of the properties of acyclic orientations in chordal graphs leads to a simple relationship: $\mathrm{AO}(T_C) = \frac{|C|!}{|\mathrm{Sep}(C)|!} \prod_{i=1}^{d} \mathrm{AO}(T_{C_i})$, where $\mathrm{AO}(T_C)$ is the number of acyclic orientations of the clique subtree T_C and where $C_1, \ldots, C_d$ are the child cliques of C in the rooted T (and $d = 0$ if C is a leaf of T). This allows us to count all acyclic orientations in time $\tilde{O}(|V(G)| + |E(G)|)$.

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 manner. Let C be the current clique we are processing, we always pick an orientation on $G[C]$ that is consistent with $G[\mathrm{Sep}(C)]$. 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. Once all cliques are processed, the resulting directed graph is just a uniformly generated random acyclic orientation. The running time is $O(\sum_{C \in T} |C|) = O(\sum_{v \in V(G)} (\deg_G(v) + 1)) = O(|E(G)|)$, assuming G is connected. The first equality follows from the fact that each v occurs in at most $\deg_G(v) + 1$ cliques. Both results are summarized in the following theorem, which also includes a more precise statement of the counting running time:

Theorem 2. *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, and a uniformly random acyclic orientation can be produced in time $O(|E(G)|)$.*

A *bipolar* (s,t)-orientation is an acyclic orientation with a unique source s and a unique sink t. We employ a similar strategy as we did for acyclic orientations. At the beginning, we construct a clique tree and randomly pick a clique C_s that contains the source s as the root clique. In order to maintain s and t as the unique source and sink, we differentiate between cliques in T that are or are not on the C_s-C_t path in T, and we recursively compute corresponding bipolar orientations of the subtrees (with some well-chosen restrictions to maintain the overall source and sink). While somewhat more complex than acyclic orientations, the structure still allows us to sample very efficiently analogously to acyclic orientations, as summarized in the following theorem:

Theorem 3. *Let G be a connected 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 conclude with sink-free orientations. Recall that for any graph there is an FPRAS counting these orientations [5] and an efficient exact uniform "sink-popping" sampler is also known [6]. Therefore, we focus on the counting problem, aiming to improve the running time compared to the FPRAS. In fact, our counting algorithm is deterministic, exact, and efficient, with (near) linear running time, as stated in this theorem:

Theorem 4. *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.*

The algorithm computes two separate quantities at every node of the clique tree: (i) NSFO(T_C), which counts orientations of $\hat{G}[T_C]$ where only sinks in Sep(C) are allowed, and (ii) ASFO(T_C, v), which counts orientations of $\hat{G}[T_C]$, where $v \in \text{Sep}(C)$ is a sink and there are no sinks in $V(\hat{G}[T_C]) - \text{Sep}(C)$. These quantities are reminiscent of the ones we used for the source-sink-free calculation, but they are significantly less involved. Due to the nature of these orientations, a single-level inclusion-exclusions is needed to compute these quantities, which allows us to run in the (near) linear running time, just as for acyclic and bipolar orientations. However, unlike for the other types of orientations, due to the negative term in the computation, this dynamic programming does not extend to a corresponding sampling algorithm. Understanding how to obtain a sampler from a dynamic programming approach combined with a single level inclusion-exclusion might help with solving our open problem related to sampling source-sink-free orientations in chordal graphs.

References

1. Agnarsson, G.: On chordal graphs and their chromatic polynomials. Mathematica Scandinavica pp. 240–246 (2003)
2. Blair, J.R., Peyton, B.: An introduction to chordal graphs and clique trees. In: Graph theory and sparse matrix computation, pp. 1–29. Springer (1993)
3. Borwein, P.B.: On the complexity of calculating factorials. Journal of Algorithms **6**(3), 376–380 (1985)
4. Brylawski, T., Oxley, J.: The tutte polynomial and its applications. Matroid applications **40**, 123–225 (1992)
5. Bubley, R., Dyer, M.: 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. pp. 248–257 (1997)
6. Cohn, H., Pemantle, R., Propp, J.: Generating a random sink-free orientation in quadratic time. arXiv preprint math/0103189 (2001)
7. Guo, H., Jerrum, M., Liu, J.: Uniform sampling through the Lovász local lemma. Journal of the ACM (JACM) **66**(3), 1–31 (2019)
8. Huber, M.: Exact sampling and approximate counting techniques. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 31–40 (1998)
9. Koller, D., Friedman, N.: Probabilistic graphical models: principles and techniques. MIT press (2009)
10. Linial, N.: Hard enumeration problems in geometry and combinatorics. SIAM Journal on Algebraic Discrete Methods **7**(2), 331–335 (1986)
11. Okamoto, Y., Uno, T., Uehara, R.: Counting the number of independent sets in chordal graphs. Journal of Discrete Algorithms **6**(2), 229–242 (2008)
12. Vandenberghe, L., Andersen, M.S.: Chordal graphs and semidefinite optimization. Foundations and Trends in Optimization **1**(4), 241–433 (2015)
13. Vertigan, D.L., Welsh, D.J.: The computational complexity of the tutte plane: the bipartite case. Combinatorics, Probability and Computing **1**(2), 181–187 (1992)
14. Welsh, D.: The tutte polynomial. Random Structures & Algorithms **15**(3-4), 210–228 (1999)
15. Welsh, D.J.: Approximate counting. Surveys in Combinatorics **241**, 287–324 (1997)
16. Wienöbst, M., Bannach, M., Liskiewicz, M.: Polynomial-time algorithms for counting and sampling markov equivalent dags. arXiv preprint arXiv:2012.09679 (2020)