Counting and Sampling Labeled Chordal Graphs in Polynomial Time

Úrsula Hébert-Johnson
University of California, Santa Barbara, CA, USA

Daniel Lokshtanov
University of California, Santa Barbara, CA, USA

Eric Vigoda
University of California, Santa Barbara, CA, USA

Abstract

We present the first polynomial-time algorithm to exactly compute the number of labeled chordal graphs on n vertices. Our algorithm solves a more general problem: given n and ω as input, it computes the number of ω-colorable labeled chordal graphs on n vertices, using $O(n^7)$ arithmetic operations. A standard sampling-to-counting reduction then yields a polynomial-time exact sampler that generates an ω-colorable labeled chordal graph on n vertices uniformly at random. Our counting algorithm improves upon the previous best result by Wormald (1985), which computes the number of labeled chordal graphs on n vertices in time exponential in n.

An implementation of the polynomial-time counting algorithm gives the number of labeled chordal graphs on up to 30 vertices in less than three minutes on a standard desktop computer. Previously, the number of labeled chordal graphs was only known for graphs on up to 15 vertices.

2012 ACM Subject Classification Theory of computation → Generating random combinatorial structures; Theory of computation → Graph algorithms analysis

Keywords and phrases Counting algorithms, graph sampling, chordal graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.58

Related Version *Full Version*: `https://arxiv.org/abs/2308.09703`

Supplementary Material *Software (Source Code)*: `https://github.com/uhebertj/chordal`
archived at `swh:1:dir:b81faedf15e1e8ebd4b68f421d2547ed39e1cd61`

Funding *Úrsula Hébert-Johnson*: Supported by NSF grant CCF-2008838.
Daniel Lokshtanov: Supported by NSF grant CCF-2008838.
Eric Vigoda: Supported by NSF grant CCF-2147094.

1 Introduction

Generating random graphs from a prescribed graph family is a fundamental task for running simulations and testing conjectures. Although generating a random labeled graph on n vertices is easy (just flip an unbiased coin for each potential edge), the first polynomial-time algorithm for generating an *unlabeled* graph uniformly at random was only given in 1987, by Wormald [36]. The algorithm of Wormald runs in polynomial time in expectation, and to the best of our knowledge, the existence of a worst-case polynomial-time sampler of random unlabeled graphs remains open.

Naturally, when we wish to sample from a specified graph family, there are many interesting families of graphs for which this problem is nontrivial, even when the graphs are labeled. For the class of labeled trees, a sampling algorithm using Prüfer sequences [27] was discovered in 1918. More recently, a fast (exact) uniform sampler was presented by Gao and Wormald for d-regular graphs with $d = o(\sqrt{n})$ in 2017 [11], and then for power-law graphs in 2018 [12]. A more general problem is the following: given an arbitrary degree sequence, generate a random graph with those specified degrees – this has been resolved for bipartite graphs [21, 3] as well as for general graphs when the maximum degree is not too large [16, 1]. See Greenhill [15] for a survey of random generation of graphs with degree constraints. For planar graphs, Bodirsky, Gröpl, and Kang presented a polynomial-time algorithm [6], which uses dynamic programming to exactly compute the number of labeled planar graphs on n vertices and generate a planar graph uniformly at random in time $\widetilde{O}(n^7)$. This was improved to $O(n^2)$ expected time by Fusy [10], using a Boltzmann sampler.

Our results fall naturally within this line of work. We consider the problem of generating a labeled *chordal* graph on n vertices uniformly at random. A graph is chordal if it has no induced cycles of length at least 4. Despite being one of the most fundamental and well-studied graph classes, prior to our work, the fastest uniform sampling algorithm for labeled chordal graphs was the exponential-time algorithm of Wormald from 1985 [35]. (To be precise, optimizing the running time of an algorithm for counting chordal graphs was not the main focus of Wormald; rather, the main goal of the paper was to determine the asymptotic number of chordal graphs with given connectivity, and the exponential-time algorithm is a corollary of these results.) Since then, various algorithmic approaches have been proposed for generating chordal graphs (e.g., [23, 32, 33, 8, 24]), but these algorithms do not come with any formal guarantees about their output distribution. In particular, [32] specifically asks for the existence of a polynomial-time algorithm to sample chordal graphs uniformly at random as an open problem. In a recent abstract, Sun and Bezáková [33] proposed a Markov chain for sampling chordal graphs, but this Markov chain comes with few mixing time guarantees.

We obtain the first polynomial (in n) time algorithm to exactly count the number of labeled chordal graphs on n vertices, as well as the first polynomial-time uniform sampling algorithm for the class of labeled chordal graphs. Our algorithm also easily extends to counting and sampling ω-*colorable* labeled chordal graphs. A graph G is ω-colorable if there exists a function $c\colon V(G) \to \{1, \ldots, \omega\}$ such that every edge $uv \in E(G)$ satisfies $c(u) \neq c(v)$.

▶ **Theorem 1.** *There is a deterministic algorithm that given positive integers n and $\omega \leq n$, computes the number of ω-colorable labeled chordal graphs on n vertices using $O(n^7)$ arithmetic operations. Moreover, there is a randomized algorithm that generates a graph uniformly at random from the set of all ω-colorable labeled chordal graphs on n vertices using $O(n^7)$ arithmetic operations.*

By the known equivalence between chromatic number, maximum clique size, and treewidth of chordal graphs [5], Theorem 1 can be reinterpreted as counting and sampling labeled chordal graphs of clique size at most ω, or treewidth at most $\omega - 1$. The running time bound of Theorem 1 is stated in terms of the number of arithmetic operations. Since there are at most 2^{n^2} labeled graphs on n vertices, the arithmetic operations need to deal with n^2-bit integers. Therefore, using the $O(n \log n)$-time algorithm for multiplying two n-bit integers [19] yields an $O(n^9 \log n)$-time upper bound for our algorithm in the RAM model.

A straightforward implementation of our counting algorithm gives the number of labeled chordal graphs on up to $n = 30$ vertices in less than three minutes on a standard desktop computer. Previously, the number of labeled chordal graphs was only known for graphs on up to 15 vertices [25]. In addition, we use our implementation to compute the number of ω-colorable labeled chordal graphs for $n \leq 12$ and $\omega \leq 12$. We chose to stop at $n = 12$ to keep the table at a reasonable size, not because of the computation time. We present the computational results in Section 4.

1.1 A Brief Survey on Chordal Graphs

The literature on chordal graphs is so vast that it would be impossible to fully do it justice. Discussions of chordal graphs in the literature go as far back as 1958 [18]. What follows is a summary of some of the most notable problems and milestones.

Many NP-hard optimization problems (such as *coloring* [13] and *maximum independent set* [9]), as well as #P-hard counting problems (such as *independent sets* [26, 4]), and many others [29], are polynomial-time solvable on chordal graphs. Chordal graphs have a wide variety of applications, including phylogeny in evolutionary biology [17, 28] and Bayesian networks in machine learning [34]. When doing Gaussian elimination on a symmetric matrix, the set of matrix entries that are nonzero for at least one time-point of the elimination process corresponds to the edge set of a chordal graph. Thus the problem of finding an ordering in which to do Gaussian elimination that minimizes the number of nonzero matrix entries can be reduced to finding a chordal supergraph of a given graph with the minimum number of edges [30]. This problem, known as *minimum fill-in*, was shown to be NP-complete by Yannakakis [37]. Chordal graphs have a central place in graph theory [7], both through their connection to treewidth [20] and through their connection to perfect graphs [14]. From an algorithms perspective, chordal graphs can be recognized in linear time [31].

An interesting and relevant result by Bender et al. [2] is that a random n-vertex labeled chordal graph is a split graph with probability $1 - o(1)$, i.e., the fraction of labeled chordal graphs that are not split is $o(1)$. This yields a simple *approximately uniform* sampler for labeled chordal graphs: simply sampling a random labeled split graph leads to an output distribution with total variation distance $o(1)$ from the uniform distribution on labeled chordal graphs. This simple sampling algorithm is unsatisfactory because it can never output a non-split chordal graph. Nevertheless, this result suggests two things: The first is that it might be possible to find a simple and efficient uniform random sampler for labeled chordal graphs. The second is that the type of chordal graphs that one usually envisions when thinking of a chordal graph (namely those with relatively small treewidth) are different from those most likely to be generated by a uniform random sampler (namely split graphs). Therefore, to generate the type of chordal graphs that one usually envisions, one should be sampling not from the set of all chordal graphs but rather from a subset, e.g., the set of all ω-colorable chordal graphs. Fortunately, Theorem 1 provides this functionality.

1.2 Methods

Our exact counting algorithm is based on dynamic programming. While clique trees and tree decompositions are never explicitly mentioned in the description of the algorithm, the intuition behind the algorithm is based on these notions. Essentially, we generate a rooted clique tree where the dynamic-programming table encodes certain properties of the graph, including how it relates to the root node of the clique tree. A *clique tree* of a graph G is a tree T together with a function f that maps each vertex of G to a connected vertex subset of T, such that for every pair u,v of vertices in G, uv is an edge in G if and only if $f(u) \cap f(v) \neq \emptyset$. It is well known that a graph G has a clique tree if and only if it is chordal [5].

The main difficulty with this approach is that different chordal graphs have different numbers of clique trees, so if we count the total number of clique trees, this will not give us an accurate count of the number of n-vertex chordal graphs. Therefore, the key idea behind our algorithm is to assign to each labeled chordal graph a unique "canonical" clique tree and to only count these canonical clique trees. The information stored in the dynamic-programming table is sufficient to ensure that every clique tree that we do count is the canonical tree of some chordal graph, and that the canonical clique tree of every chordal graph is counted. As it turns out, the best way to phrase our algorithm is not in terms of clique trees at all, but rather in terms of an (essentially) equivalent notion that we call an "evaporation sequence." An evaporation sequence is closely related to the standard notion of a perfect elimination ordering (PEO) of a chordal graph. A *simplicial* vertex is a vertex whose neighborhood is a clique, and a PEO is an ordering of the vertices such that each is simplicial in the current induced subgraph, if we delete the vertices in that order (see Section 3.1 for more details). An evaporation sequence is a type of "canonical" PEO: at each step, we remove *all* simplicial vertices from the current subgraph, rather than making an arbitrary choice of a single simplicial vertex. We say that all of the simplicial vertices evaporate at time 1. Next, all of the vertices that become simplicial once the first set of simplicial vertices has been removed are said to evaporate at time 2, and so on. It is easy to see that every labeled chordal graph has a unique evaporation sequence, and that this sequence does not depend on the labeling of the vertices.

Therefore, the number of chordal graphs on n vertices is the sum over all evaporation sequences of the number of labeled chordal graphs with that evaporation sequence. While different evaporation sequences correspond to different numbers of chordal graphs, because this number is independent of the labels, we can simply guess the *number* x of vertices that evaporate at any given time, and then without loss of generality assign the labels $1, \dots, x$ to those vertices.

In our dynamic-programming algorithm, the recursive subproblems deal with counting *rooted* clique trees. In this context, the root of a clique tree is the set of vertices that evaporate last. Just as we would like to "force" a set of nodes to be in the root of the tree, we will sometimes want to force a set of nodes to evaporate last. This is done using what we call an *exception set*, i.e., a set of vertices that do not evaporate even if they are simplicial.

The random sampling algorithm in Theorem 1 follows from our counting algorithm using the standard sampling-to-counting reduction of [22].

1.3 Overview of the Paper

Our dynamic-programming algorithm, including the associated recurrences, is presented in Section 3. For a glimpse of the proof of correctness, one can find the proof of the first recurrence (the reduction to counting *connected* chordal graphs) in Section 3.4. The complete proof of correctness, as well as the details of how to obtain the random sampling algorithm using the counting algorithm, can be found in the full version of the paper.

2 Preliminaries

Our algorithm counts vertex-labeled chordal graphs. For simplicity of notation, we assume the vertex set of each graph is a subset of $\mathbb{N} = \{1, 2, 3, \dots\}$, which allows the labels to also serve as the names of the vertices. For example, we will speak of the vertex $5 \in V(G)$ rather than a vertex v with label 5.

▶ **Definition 2.** *A* labeled graph *is a pair $G = (V, E)$, where the vertex set V is a finite subset of $\mathbb{N}$ and the edge set E is a set of two-element subsets of V.*

Henceforth, we implicitly assume *all graphs that we consider are labeled graphs.* For nonnegative integers n, we use the notation $[n] := \{1, 2, \dots, n\}$. Intervals of integers will often appear in our algorithm as the vertex set of a graph or subgraph, so we also define

$$[a, b] := \{a, a + 1, \dots, b\}$$

for nonnegative integers a, b. If $b < a$, then $[a, b] = \emptyset$.

▶ **Definition 3.** *Let $A = \{a_1, \dots, a_r\}$ and $B = \{b_1, \dots, b_r\}$ be finite subsets of $\mathbb{N}$ such that $|A| = |B|$, where the elements a_i and b_i are listed in increasing order. We define $\phi(A, B)\colon A \to B$ as the bijection that maps a_i to b_i for all $i \in [r]$.*

▶ **Definition 4.** *Let G_1, G_2 be two graphs, and suppose $C \coloneqq V(G_1) \cap V(G_2)$ is a clique in both G_1 and G_2. When we say we* glue G_1 and G_2 together at C *to obtain G, this indicates that G is the union of G_1 and G_2: the vertex set is $V(G) = V(G_1) \cup V(G_2)$, and the edge set is $E(G) = E(G_1) \cup E(G_2)$.*

For a graph G and a vertex subset $S \subseteq V(G)$, $G[S]$ denotes the induced subgraph on the vertices of S. For a vertex $v \in V(G)$, we denote the neighborhood of v in G by $N_G(v)$, or by $N(v)$ if the graph is clear from the context. For $S \subseteq V(G)$, the open neighborhood of S is denoted by

$$N_G(S) \coloneqq \{v \in V(G) \setminus S : uv \in E(G) \text{ for some } u \in S\}$$

and the closed neighborhood of S is denoted by $N_G[S] \coloneqq S \cup N_G(S)$, or simply $N(S)$ and $N[S]$, respectively. For $S, T \subseteq V(G)$, we say S *sees all of* T if $T \subseteq N(S)$.

3 Counting labeled chordal graphs

3.1 The evaporation sequence

▶ **Definition 5.** *A vertex v in a graph G is* simplicial *if $N(v)$ forms a clique.*

A *perfect elimination ordering* of a graph G is an ordering $v_1, \dots, v_n$ of the vertices of G such that for all $i \in [n]$, v_i is simplicial in the subgraph induced by the vertices $v_i, \dots, v_n$. It is well known that a graph is chordal if and only if it has a perfect elimination ordering [5]. For our counting algorithm, we define the notion of the evaporation sequence of a chordal graph, which can be viewed as a canonical version of the perfect elimination ordering. In the evaporation sequence, rather than making an arbitrary choice for which of the simplicial vertices in G will go first in the ordering, we place the set of all simplicial vertices as the first item in the sequence. As an example, if G is a tree, then the set of all simplicial vertices would be exactly the leaves of G.

To build the evaporation sequence, we use the fact that given a chordal graph G, if we repeatedly remove all simplicial vertices, then eventually no vertices remain. By observing at which time step each vertex is deleted, we obtain a partition of $V(G)$, which allows us to classify and understand the structure of G. In our algorithm, it will also be useful to set aside a set of exceptional vertices which are never deleted, even if they are simplical.

To formalize this, suppose we are given a chordal graph G and a vertex subset $X \subseteq V(G)$. We define the *evaporation sequence* of G with *exception set* X as follows: If $X = V(G)$, then the evaporation sequence of G is the empty sequence. If $X \subsetneq V(G)$, then let $\widetilde{L}_1$ be the set of all simplicial vertices in G, and let $L_1 = \widetilde{L}_1 \setminus X$. Suppose $L_2, \dots, L_t$ is the evaporation sequence of $G \setminus L_1$ (with exception set X). Then $L_1, L_2, \dots, L_t$ is the evaporation sequence of G.

For this definition to make sense, there is one caveat: we must choose X so that all vertices outside of X eventually evaporate. For example, $X = \emptyset$ is always a valid choice since every chordal graph contains a simplicial vertex [5]. Without this assumption, we could potentially reach a point where $X \subsetneq V(G)$ but X contains all of the simplicial vertices of G, in which case the evaporation sequence would not be well-defined (we never reach the base case $X = V(G)$). In our algorithm, we will always choose a valid X such that all other vertices eventually evaporate.

If the evaporation sequence $L_1, L_2, \dots, L_t$ of G has length t, then we say G *evaporates* at time t with *exception set* X, and t is called the *evaporation time*. We define $L_G(X) := L_t$ to be the last set in the evaporation sequence of G, and we let $L_G(X) = \emptyset$ if the sequence is empty. Similarly, we define the evaporation time of a vertex subset: Suppose G has evaporation sequence $L_1, L_2, \dots, L_t$ with exception set X, and suppose $S \subseteq V(G) \setminus X$ is a nonempty vertex subset. Let t_S be the largest index i such that $L_i \cap S \neq \emptyset$. We say S *evaporates* at time t_S in G with exception set X.

3.2 Setup for the counting algorithm

Given positive integers n and ω, we wish to count the number of ω-colorable chordal graphs on n vertices. This can easily be reduced to the problem of counting *connected* ω-colorable chordal graphs on at most n vertices (see Lemma 17 in Section 3.4). Therefore, our main focus is to describe the following algorithm:

▶ **Theorem 6.** *There is an algorithm that given $n \in \mathbb{N}$, computes the number of ω-colorable labeled connected chordal graphs G with vertex set $[n]$ using $O(n^7)$ arithmetic operations.*

We first give an overview of this algorithm and describe the various dynamic-programming tables (Definition 7). Next, we describe the recurrences in detail in Section 3.3. In Section 3.4, we show that the counting portion of Theorem 1 (counting chordal graphs) follows from Theorem 6 (counting connected chordal graphs). For the complete proof of Theorem 6, see the full version of the paper.

Algorithm overview. To count ω-colorable connected chordal graphs G, we classify these graphs based on the behavior of their evaporation sequence. We make use of several *counter functions* (these are our dynamic-programming tables), each of which keeps track of the number of chordal graphs in a particular subclass. The arguments of the counter functions tell us the number of vertices in the graph, the evaporation time, the size of the exception set X, the size of the last set of simplicial vertices $L_G(X)$, etc. Initially, we consider all possibilities for the evaporation time of G with exception set $X = \emptyset$. Then, using several of our recursive formulas, we reduce the number of vertices by dividing up the graph into smaller subgraphs and counting the number of possibilities for each subgraph. As we do so, the exception set X increases in size. When we consider these various subgraphs, we also make sure that in each subgraph, the maximum clique size is at most ω. In the end, the algorithm understands the possibilities for the entire graph, including the cliques that make up the very first set in its evaporation sequence.

The purpose of the exception set is to allow us to restrict to smaller subgraphs without distorting the evaporation behavior of the graph. For example, suppose we wish to count the number of connected chordal graphs on n vertices that evaporate at time t, such that the vertices $1, 2, \dots, \ell$ make up the last set to evaporate, i.e., $L_G(\emptyset) = [\ell]$. Let $L = [\ell]$. One subproblem of interest would be to count the number of possibilities for the first connected component of $G \setminus L$. Formally, we count the number of possibilities for $G' := G[L \cup C]$, where C is the connected component of $G \setminus L$ that contains the vertex $\ell + 1$. For each possible number of vertices in G', we make a recursive call to count the number of possible subgraphs G' of that size. However, if we were to restrict to G' with a still-empty exception set, then the evaporation time of G' alone could be much less than the evaporation time of $V(G')$ in G. Indeed, there may be vertices in $G \setminus G'$ adjacent to L that prevent L from evaporating before time t, so when we restrict to the subgraph G', L would now evaporate too soon. This would cause a cascading effect, causing vertices near L to evaporate as well, and changing the entire evaporation sequence of G'. To resolve this, we add the vertices of L to the exception set to preserve the evaporation behavior of G'.

The list of counter functions is given in Definition 7. As shown below, the number of ω-colorable connected chordal graphs on n vertices is the sum of various calls to the fourth function $\tilde{g}_1$, since $\tilde{g}_1(t, 0, n)$ is the number of ω-colorable connected chordal graphs on n vertices that evaporate at time t with empty exception set. To remember the names of these counter functions, one can think of them as follows: The g-functions keep track of the size of the exception set X, but these do not have information about the size of $L_G(X)$. The f-functions have an additional argument ℓ, which is the size of $L_G(X)$. As a mnemonic, one can say that g stands for "glued" and f stands for "free." In the g-functions, X is the "root," and all of the vertices of X are glued, in the sense that they cannot evaporate. In the f-functions, $X \cup L_G(X)$ is the "root," and some of the vertices in the root are free, since the vertices in $L_G(X)$ are allowed to evaporate.

▶ **Definition 7.** *The following functions count particular subclasses of chordal graphs. Unless stated otherwise, the arguments t, x, ℓ, k, z are nonnegative integers.*

1. *$g(t, x, k, z)$ is the number of ω-colorable connected chordal graphs G with vertex set $[x + k]$ that evaporate in time at most t with exception set $X = [x]$, where X is a clique, with the following property: every connected component of $G \setminus X$ (if any) has at least one neighbor in $X \setminus [z]$.* ***Domain:*** *$t \geq 0$, $x \geq 1$, $z < x$.*
2. *$\tilde{g}(t, x, k, z)$ is the same as $g(t, x, k, z)$, except every connected component of $G \setminus X$ (if any) evaporates at time* exactly *t in G.* Note: A graph with $V(G) = X$ would be counted because in that case, $\tilde{g}$ is the same as g. ***Domain:*** *$t \geq 1$, $x \geq 1$, $z < x$.*
3. *$\tilde{g}_p(t, x, k, z)$ is the same as $\tilde{g}(t, x, k, z)$, except no connected component of $G \setminus X$ sees all of X.* ***Domain:*** *$t \geq 1$, $x \geq 1$, $z < x$.*
4. *$\tilde{g}_1(t, x, k)$ and $\tilde{g}_{\geq 2}(t, x, k)$ are the same as $\tilde{g}(t, x, k, z)$, except every connected component of $G \setminus X$ sees all of X (hence we no longer require every component of $G \setminus X$ to have a neighbor in $X \setminus [z]$), and furthermore, for $\tilde{g}_1$ we require that $G \setminus X$ has exactly one connected component, and for $\tilde{g}_{\geq 2}$ we require that $G \setminus X$ has at least two components.* ***Domain for $\tilde{g}_1$:*** *$t \geq 1$, $x \geq 0$.* ***Domain for $\tilde{g}_{\geq 2}$:*** *$t \geq 1$, $x \geq 1$.*
5. *$f(t, x, \ell, k)$ is the number of ω-colorable connected chordal graphs G with vertex set $[x + \ell + k]$ that evaporate at time* exactly *t with exception set $X = [x]$, such that $G \setminus X$ is connected, $L_G(X) = [x + 1, x + \ell]$, and $X \cup L_G(X)$ is a clique.* ***Domain:*** *$t \geq 1$, $x \geq 0$, $\ell \geq 1$.*
6. *$\tilde{f}(t, x, \ell, k)$ is the same as $f(t, x, \ell, k)$, except every connected component of $G \setminus (X \cup L_G(X))$ evaporates at time* exactly *$t - 1$ in G, and there exists at least one such component, i.e., $X \cup L_G(X) \subsetneq V(G)$.* ***Domain:*** *$t \geq 2$, $x \geq 0$, $\ell \geq 1$.*
7. *$\tilde{f}_p(t, x, \ell, k)$ is the same as $\tilde{f}(t, x, \ell, k)$, except no connected component of $G \setminus (X \cup L_G(X))$ sees all of $X \cup L_G(X)$.* ***Domain:*** *$t \geq 2$, $x \geq 0$, $\ell \geq 1$.*
8. *$\tilde{f}_p(t, x, \ell, k, z)$ is the same as $\tilde{f}_p(t, x, \ell, k)$, except every connected component of $G \setminus (X \cup L_G(X))$ has at least one neighbor in $(X \cup L_G(X)) \setminus [z]$.* ***Domain:*** *$t \geq 2$, $x \geq 0$, $\ell \geq 1$, $z \leq x$.*

3.3 Recurrences for the counting algorithm

We implicitly assume *all graphs in this section are connected and ω-colorable.* For $k \in \mathbb{N}$, let $c(k)$ denote the number of ω-colorable connected chordal graphs with vertex set $[k]$. To compute $c(n)$, we first consider all possibilities for the evaporation time. Initially, the exception set is empty. We observe that $\tilde{g}_1(t, 0, n)$ is the number of (connected, ω-colorable) chordal graphs with vertex set $[n]$ that evaporate at time exactly t with empty exception set.

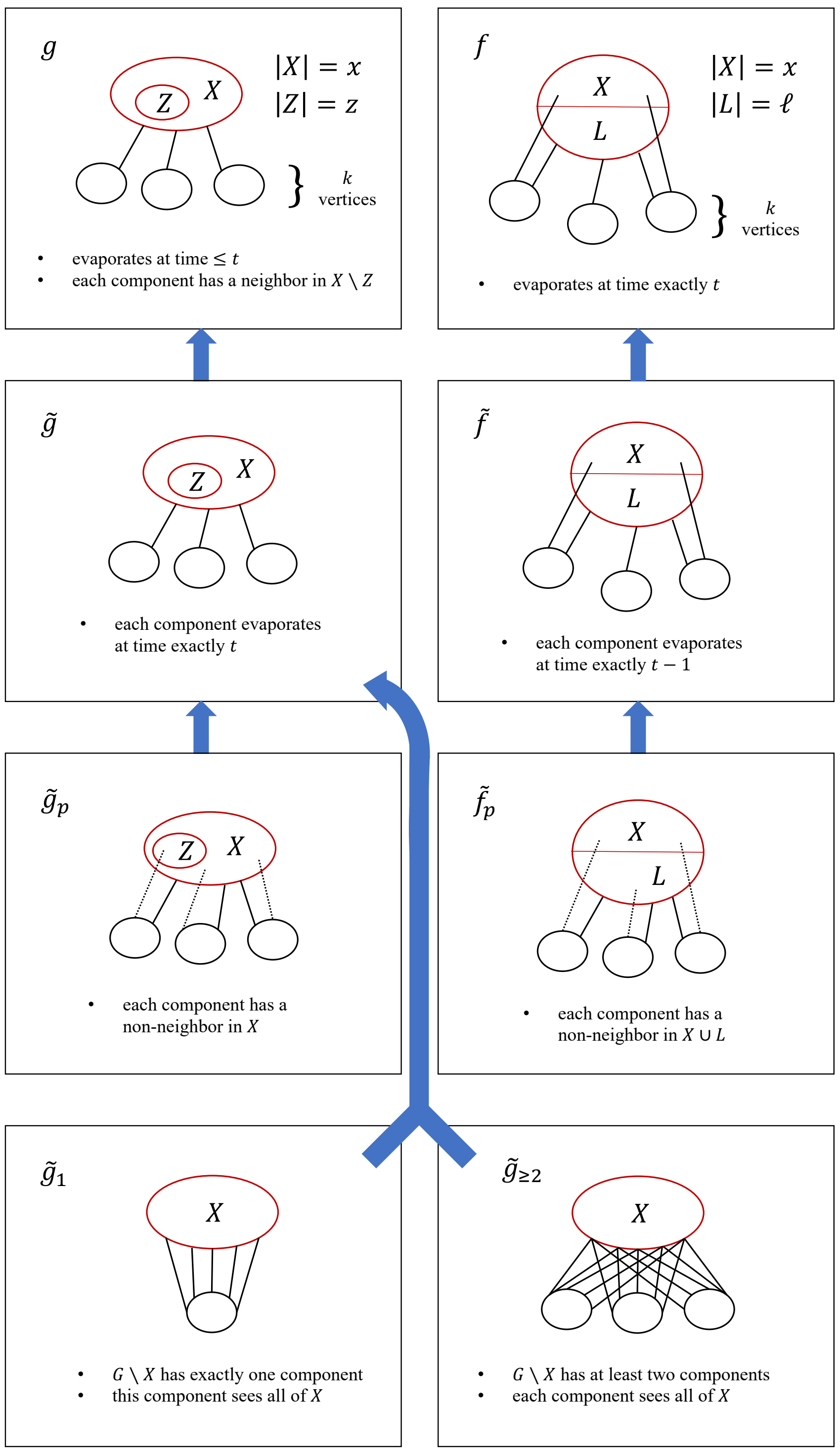
g
$|X| = x$
$|Z| = z$
Z
X
k vertices
• evaporates at time $\leq t$
• each component has a neighbor in $X \setminus Z$
f
$|X| = x$
$|L| = \ell$
X
L
k vertices
• evaporates at time exactly t
$\tilde{g}$
Z
X
• each component evaporates at time exactly t
$\tilde{f}$
X
L
• each component evaporates at time exactly $t-1$
$\tilde{g}_p$
Z
X
• each component has a non-neighbor in X
$\tilde{f}_p$
X
L
• each component has a non-neighbor in $X \cup L$
$\tilde{g}_1$
X
• $G \setminus X$ has exactly one component
• this component sees all of X
$\tilde{g}_{\geq 2}$
X
• $G \setminus X$ has at least two components
• each component sees all of X

Figure 1 The counter functions. Here $L = L_G(X)$ and $Z = [z]$. An arrow from one function to another, say from $\tilde{g}$ to g, indicates that the definition of $\tilde{g}$ is the same as that of g, except where indicated otherwise. The drawing of $\tilde{f}_p$ represents $\tilde{f}_p(t, x, \ell, k)$. The function $\tilde{f}_p(t, x, \ell, k, z)$ is similar but also keeps track of the argument z.

Therefore,

$$c(n) = \sum_{t=1}^{n} \tilde{g}_1(t, 0, n).$$

We compute the necessary values of $\tilde{g}_1$ by evaluating the following recurrences top-down using memoization. We take this approach rather than bottom-up dynamic programming to simplify the description slightly, since by memoizing we do not need to specify in what order the entries of the various dynamic-programming tables are computed. We simply compute each value of the counter functions as needed. For the following recurrences, let $X = [x]$ according to the current value of the argument x, and let $L = L_G(X)$.

To compute $\tilde{g}_1(t, 0, n)$, the number of chordal graphs that evaporate at time exactly t, we consider all possibilities for the size ℓ of L. Once the size ℓ is given, there are $\binom{n}{\ell}$ possibilities for the label set of L. Recall that f counts the number of chordal graphs where L is fixed and evaporates at time t, and $G \setminus X$ is connected. Formally, we have the following recurrence – the first time this is used, the arguments are t, $x = 0$ and $k = n$.

▶ **Lemma 8.** *For $\tilde{g}_1$, we have*

$$\tilde{g}_1(t, x, k) = \sum_{\ell=1}^{k} \binom{k}{\ell} f(t, x, \ell, k - \ell).$$

The proof of Lemma 8, along with the proofs of all of the other recurrences, can be found in the full version of the paper. To see the intuition behind Lemma 8, recall that in the definition of $f(t, x, \ell, k)$ we require a specific label set for $L_G(X)$, namely $[x+1, x+\ell]$. If we were to replace that requirement with $L_G(X) = L'$ for any other subset L' of $[x+1, x+\ell+k]$ of size ℓ, this would not change the value of $f(t, x, \ell, k)$. Therefore, in the recurrence for $\tilde{g}_1$, it is sufficient to compute $f(t, x, \ell, k - \ell)$ and multiply by $\binom{k}{\ell}$, rather than computing $\binom{k}{\ell}$ distinct counter functions.

For f, to count chordal graphs where L is fixed and evaporates at time t, and $G \setminus X$ is connected, we consider all possibilities for the set of labels that appear in components of $G \setminus (X \cup L)$ that evaporate at time exactly $t - 1$. Recall that $\tilde{f}$ counts the number of chordal graphs where L is fixed and evaporates at time t, $G \setminus X$ is connected, and all components of $G \setminus (X \cup L)$ evaporate at time exactly $t - 1$. For each possible k' (the size of this label set), $\tilde{f}$ allows us to count the number of possibilities for the subgraph G_1 consisting of $X \cup L$ and all components of $G \setminus (X \cup L)$ that evaporate at time exactly $t - 1$, and g allows us to count the number of possibilities for the subgraph G_2 consisting of $X \cup L$ and all other components.

▶ **Lemma 9.** *For f, we have*

$$f(t, x, \ell, k) = \sum_{k'=1}^{k} \binom{k}{k'} \tilde{f}(t, x, \ell, k') g(t - 2, x + \ell, k - k', x).$$

When f is called for the first time in the initial steps of the algorithm, this is the first moment when X becomes nonempty, since at this point we are restricting to a subgraph with fewer than n vertices. When we restrict to the subgraph G_2, we want to ensure that its vertices have the same evaporation behavior as they did in G. In particular, we need to ensure that the vertices of L do not evaporate too soon, since their presence may be essential for preventing other vertices from evaporating. For this reason, we let $X \cup L$ be the exception set for G_2. For G_1, the exception set is simply X because the components that evaporate at time exactly $t - 1$ are still present in G_1, preventing the vertices of L from evaporating before time t.

For the subgraph G_2, now that all of L has been pushed into the exception set, we no longer have information about the argument ℓ since the last set of simplicial vertices in G_2 evaporates further back in time. This is why we call g rather than f to count the possibilities for G_2. In fact, $G_2 \setminus [x+\ell]$ might not even be connected, which is required by f. Finally, the fourth argument of g indicates that every connected component of $G_2 \setminus (X \cup L)$ has at least one neighbor in L. This ensures that $G \setminus X$ is connected.

For g, to count chordal graphs that evaporate in time at most t, we consider all possibilities for the set of labels that appear in connected components of $G \setminus X$ that evaporate at time exactly t. Recall that $\tilde{g}$ counts the number of chordal graphs where all connected components of $G \setminus X$ evaporate at time exactly t.

▶ **Lemma 10.** *For g, we have*

$$g(t,x,k,z) = \sum_{k'=0}^{k} \binom{k}{k'} \tilde{g}(t,x,k',z) g(t-1,x,k-k',z).$$

For $\tilde{g}$, to count chordal graphs where all connected components of $G \setminus X$ evaporate at time exactly t, we consider all possible label sets for the component C of $G \setminus X$ that contains the lowest label not in X. The constraint $x' \geq 1$ ensures that G is connected. We also subtract all ways of selecting x' elements from $[z]$ to ensure that $N(C)$ is not entirely contained in $[z]$.

▶ **Lemma 11.** *For $\tilde{g}$, we have*

$$\tilde{g}(t,x,k,z) = \sum_{k'=1}^{k} \sum_{x'=1}^{x} \left(\binom{x}{x'} - \binom{z}{x'} \right) \binom{k-1}{k'-1} \tilde{g}_1(t,x',k') \tilde{g}(t,x,k-k',z).$$

We subtract 1 in the binomial coefficient $\binom{k-1}{k'-1}$ because the label set for C always contains the lowest non-X label, along with $k'-1$ other labels.

For $\tilde{f}$, we need to count chordal graphs where L is fixed and evaporates at time t, $G \setminus X$ is connected, and all components of $G \setminus (X \cup L)$ evaporate at time exactly $t-1$. The number of such graphs in which zero components see all of $X \cup L$ is $\tilde{f}_p(t,x,\ell,k)$. Now if there is at least one all-seeing component, then we break this down into two further cases: either exactly one component sees all of $X \cup L$, or at least two components see all of $X \cup L$. Recall that $\tilde{f}_p$ counts the number of chordal graphs where L is fixed and evaporates at time t, all components of $G \setminus (X \cup L)$ evaporate at time exactly $t-1$, and no component sees all of $X \cup L$. Also, recall that $\tilde{g}_{\geq 2}$ counts the number of chordal graphs where all components of $G \setminus X$ evaporate at time exactly t, every component sees all of X, and there are at least two such components. In the first (resp. second) case, $\tilde{g}_1$ (resp. $\tilde{g}_{\geq 2}$) corresponds to the all-seeing component(s), and $\tilde{f}_p$ (resp. $\tilde{g}_p$) corresponds to the remaining components.

▶ **Lemma 12.** *For $\tilde{f}$, we have*

$$\tilde{f}(t,x,\ell,k) = \tilde{f}_p(t,x,\ell,k) + \sum_{k'=1}^{k} \binom{k}{k'} \tilde{g}_1(t-1,x+\ell,k') \tilde{f}_p(t,x,\ell,k-k') + \sum_{k'=1}^{k} \binom{k}{k'} \tilde{g}_{\geq 2}(t-1,x+\ell,k') \tilde{g}_p(t-1,x+\ell,k-k',x).$$

The above cases are relevant because if at least two components of $G \setminus (X \cup L)$ see all of $X \cup L$, then this prevents the vertices of L from evaporating before time t. Indeed, each vertex $u \in L$ has a neighbor in each of those two components, meaning u has two non-adjacent neighbors. Otherwise, if there is at most one such component, then the neighborhoods of the remaining components of $G \setminus (X \cup L)$ must together cover L to ensure that L does not evaporate until time t. In that case, for each vertex $u \in L$ that is covered by a proper-subset neighborhood $N(C)$ of a component C, u has a neighbor $v \in C$ as well as a neighbor $w \in (X \cup L) \setminus N(C)$, and v and w are non-adjacent.

The reason we require $G \setminus X$ to be connected in the definition of f (rather than just requiring G to be connected) can be seen from the recurrence for $\tilde{f}$. Since in the first sum over k' we only wish to consider graphs with exactly one all-seeing component, in the definition of $\tilde{g}_1$ we require $G \setminus X$ to be connected. The recurrence for $\tilde{g}_1$ depends on f, so this carries over into requiring $G \setminus X$ to be connected in the definition of f. This explains the need for the argument z (for example, in g): as mentioned above, keeping track of z lets us ensure that $G \setminus X$ is connected in all graphs counted by f.

For $\tilde{g}_{\geq 2}$, to count chordal graphs where all components of $G \setminus X$ evaporate at time exactly t, every component sees all of X, and there are at least two such components, we consider all possibilities for the label set of the component that contains the lowest label not in X. For the remaining components, there is either exactly one of them, or at least two.

▶ **Lemma 13.** *For $\tilde{g}_{\geq 2}$, we have*

$$\tilde{g}_{\geq 2}(t,x,k) = \sum_{k'=1}^{k-1} \binom{k-1}{k'-1} \tilde{g}_1(t,x,k') \Big(\tilde{g}_1(t,x,k-k') + \tilde{g}_{\geq 2}(t,x,k-k') \Big).$$

For $\tilde{g}_p$, to count chordal graphs where all components of $G \setminus X$ evaporate at time exactly t and no component sees all of X, we proceed as we did for $\tilde{g}$, except we require $x' < x$ rather than $x' \leq x$.

▶ **Lemma 14.** *For $\tilde{g}_p$, we have*

$$\tilde{g}_p(t,x,k,z) = \sum_{k'=1}^{k} \sum_{x'=1}^{x-1} \left(\binom{x}{x'} - \binom{z}{x'} \right) \binom{k-1}{k'-1} \tilde{g}_1(t,x',k') \tilde{g}_p(t,x,k-k',z).$$

For $\tilde{f}_p$, to count chordal graphs where L is fixed and evaporates at time t, all components of $G \setminus (X \cup L)$ evaporate at time exactly $t-1$, and no component sees all of $X \cup L$, we first declare that no component can see only into X (since $G \setminus X$ is connected).

▶ **Lemma 15.** *We have $\tilde{f}_p(t,x,\ell,k) = \tilde{f}_p(t,x,\ell,k,x)$.*

The following recurrence for $\tilde{f}_p$ counts the number of such graphs in which every component of $G \setminus (X \cup L)$ has at least one neighbor in $(X \cup L) \setminus [z]$. On the first reading, one can skip the two "otherwise" cases in Lemma 16. In this lemma, we consider all possibilities for the label set of the component C of $G \setminus (X \cup L)$ that contains the lowest label not in $X \cup L$. Additionally, we consider all possibilities for the size x' of $N(C) \cap X$ and the size ℓ' of $N(C) \cap L$, and we consider all possibilities for their respective label sets. If $\ell' > 0$, then $N(C)$ is automatically not contained in $[z]$ since $z \leq x$, so there are $\binom{x}{x'}$ possible label sets for $N(C) \cap X$.

The intuition behind the two "otherwise" cases is as follows. If $\ell' = 0$, then we must subtract $\binom{z}{x'}$ from the number of possible label sets for $N(C) \cap X$ to ensure that $N(C) \subsetneq [z]$. If $\ell' = \ell$, then all of the vertices of L have now been pushed into the exception set, so the evaporation time of the subgraph formed from the remaining components is $t-1$. In this case, we call $\tilde{g}_p$ since we no longer know the size of the last set of simplicial vertices.[1]

[1] One might wonder whether we depart from the domain of $\tilde{g}_p$ in the term $\tilde{g}_p(t-1, x+\ell', k-k', z)$, since $x + \ell' = z$ when $\ell' = 0$ and $x = z$. However, if $\ell' = 0$ and $x = z$, then we observe that $\binom{x}{x'} - \binom{z}{x'} = 0$. Thus, for this value of ℓ', we do not need to evaluate the calls to $\tilde{g}_1$, $\tilde{f}_p$, and $\tilde{g}_p$.

▶ **Lemma 16.** *For $\tilde{f}_p(t, x, \ell, k, z)$, we have*

$$\tilde{f}_p(t,x,\ell,k,z) = \sum_{k'=1}^{k} \sum_{\substack{0 \le x' \le x \\ 0 \le \ell' \le \ell \\ 0 < x'+\ell' < x+\ell}} \binom{k-1}{k'-1}\binom{\ell}{\ell'} \tilde{g}_1(t-1, x'+\ell', k') \cdot \begin{cases} \binom{x}{x'} & \text{if } \ell' > 0 \\ \binom{x}{x'} - \binom{z}{x'} & \text{otherwise} \end{cases}$$
$$\cdot \begin{cases} \tilde{f}_p(t, x+\ell', \ell-\ell', k-k', z) & \text{if } \ell' < \ell \\ \tilde{g}_p(t-1, x+\ell', k-k', z) & \text{otherwise.} \end{cases}$$

The base cases are as follows. We reach the base case for g when $t = 0$:

$$g(0,x,k,z) = \begin{cases} 1 & \text{if } k = 0 \\ 0 & \text{if } k > 0. \end{cases}$$

For $\tilde{g}$ and $\tilde{g}_p$, we have $\tilde{g}(t, x, 0, z) = 1$ and $\tilde{g}_p(t, x, 0, z) = 1$ when $k = 0$. For, $\tilde{g}_1$ we observe that $\tilde{g}_1(t, x, k) = 0$ if $t = 0$ or $k = 0$. Similarly, for $\tilde{g}_{\ge 2}$ we have $\tilde{g}_{\ge 2}(t, x, k) = 0$ if $t = 0$ or $k = 0$. We reach the base case for f when $x + \ell > \omega$, $t = 1$, or $k = 0$. If $x + \ell > \omega$, then $f(t, x, \ell, k) = 0$. Remarkably, this is the only place where ω appears in the algorithm. If $x + \ell \le \omega$, then we have

$$f(1,x,\ell,k) = \begin{cases} 1 & \text{if } k = 0 \\ 0 & \text{otherwise.} \end{cases}$$

If $x + \ell \le \omega$ and $t \ge 2$, then $f(t, x, \ell, 0) = 0$. For $\tilde{f}$, we have $\tilde{f}(t, x, \ell, k) = 0$ if $t = 1$ or $k = 0$. Similarly, for $\tilde{f}_p$ we have $\tilde{f}_p[t, x, \ell, k, z] = 0$ if $t = 1$ or $k = 0$. For the version of $\tilde{f}_p$ without the fifth argument z, we do not need a base case since we always immediately call $\tilde{f}_p$ with z.

The control flow formed by these recurrences is shown in Figure 2. The algorithm terminates because either the value of t or the number of vertices in the graph (i.e., $x + k$ or $x + \ell + k$) decreases each time we return to the same function. For the running time, this is dominated by the arithmetic operations needed to compute $\tilde{f}_p$. The recurrence for $\tilde{f}_p$ involves a triple summation, and there are five arguments, so a naive implementation uses $O(n^8)$ arithmetic operations. However, in the full version of the paper, we show that the running time can in fact be improved to $O(n^7)$ arithmetic operations.

3.4 Proof of Theorem 1 (counting)

In this section, we prove the counting portion of Theorem 1 using Theorem 6. (See the full version of the paper for the proof of the sampling portion of Theorem 1.) In other words, we describe an algorithm to count chordal graphs, assuming we have an algorithm to count connected chordal graphs. Theorem 6 – counting connected chordal graphs – is proved in the full version of the paper.

For $k \in \mathbb{N}$, let $a(k)$ denote the number of ω-colorable chordal graphs with vertex set $[k]$. Recall that $c(k)$ is the number of ω-colorable connected chordal graphs with vertex set $[k]$.

▶ **Lemma 17.** *The number of ω-colorable chordal graphs with vertex set $[n]$ is given by*

$$a(n) = \sum_{k=1}^{n} \binom{n-1}{k-1} c(k) a(n-k)$$

for all $n \in \mathbb{N}$.

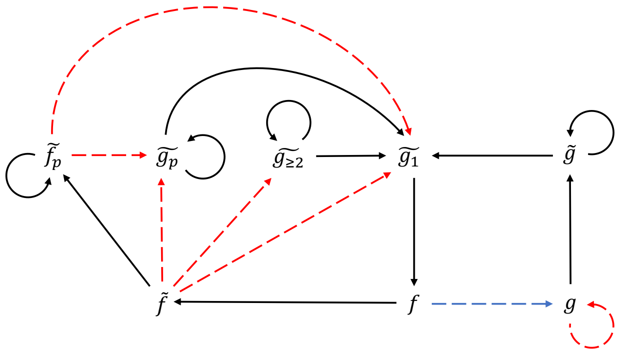

Figure 2 The control flow of the counting algorithm. An arrow from f to g indicates that the recursive formula for f depends on g. The arrow is red (and dashed) if t decreases by 1, blue (and dashed) if t decreases by 2, and black if t does not change. The black self-loops do not cause an infinite loop because in those recursive calls, the number of vertices decreases.

Proof. Suppose G is an ω-colorable chordal graph with vertex set $[n]$. Let G_1 be the graph formed by the connected component of G that contains the label 1, and let G' be the graph formed by all other connected components of G (which can potentially be empty). Let C be the set of labels that appear in G_1, let $k = |C|$, and let $D = [n] \setminus C$, i.e., D is the set of labels that appear in G'. Now relabel G_1 by applying $\phi(C, [k])$ to the labels in C, and relabel G' by applying $\phi(D, [n-k])$ to the labels in D. (Recall that $\phi(A, B)$ is defined in Section 2.) We can see that G_1 is now a connected ω-colorable chordal graph with vertex set $[k]$, and G' is a connected ω-colorable chordal graph with vertex set $[n-k]$. The map that takes any chordal graph G to the resulting pair (G_1, G') is injective since $\phi(C, [k])$ and $\phi(D, [n-k])$ are both bijections. Therefore, $a(n)$ is at most the number of possible triples (G_1, G', C), which is given by the summation above.

To see that $a(n)$ is bounded below by the same summation, suppose we are given $1 \leq k \leq n$, a connected ω-colorable chordal graph G_1 with vertex set $[k]$, a chordal ω-colorable graph G' with vertex set $[n-k]$, and a subset $C \subseteq [n]$ of size k that contains 1. Let $D = [n] \setminus C$. We construct an ω-colorable chordal graph G with vertex set $[n]$ as follows: Relabel G_1 by applying $\phi([k], C)$ to its label set, and relabel G' by applying $\phi([n-k], D)$ to its label set. Now let G be the union of G_1 and G' (by taking the union of the vertex sets and the edge sets). The map that takes the triple (G_1, G', C) to the resulting graph G is injective, so $a(n)$ is at least the summation above, as desired. ◀

Lemma 17 directly gives a dynamic-programming algorithm to compute the number of ω-colorable chordal graphs with vertex set $[n]$, given n as input. First, by Theorem 6, we can compute $c(k)$ for all $k \in [n]$ at a cost of $O(n^7)$ arithmetic operations. Next, we use the recurrence in Lemma 17 to compute $a(n)$ at a cost of $O(n^2)$ arithmetic operations. For the base case, we observe that $a(0) = 1$. Therefore, we have an algorithm to count ω-colorable chordal graphs on n vertices using $O(n^7)$ arithmetic operations.

4 Implementation of the counting algorithm

An implementation of the counting algorithm in C++ can be successfully run for inputs as large as $n = 30$ in about 2.5 minutes on a standard desktop computer.[2] Previously, the number of labeled chordal graphs was only known up to $n = 15$. Table 1 shows the number of

[2] Our implementation is available on GitHub at `https://github.com/uhebertj/chordal`.

connected chordal graphs on n vertices for $n \leq 30$, with the chromatic number unrestricted. Table 2 shows the number of ω-colorable connected chordal graphs on n vertices for various values of n and ω.

Table 1 Numbers of labeled connected chordal graphs on n vertices.

$c(n)$	n
1	1
1	2
4	3
35	4
541	5
13302	6
489287	7
25864897	8
1910753782	9
193328835393	10
26404671468121	11
4818917841228328	12
1167442027829857677	13
374059462390709800421	14
158311620026439080777076	15
88561607724193506845709239	16
65629642803250494352023169033	17
64646285130595946195244365518454	18
84997214469704246545711429635276299	19
149881423568752945444616261913109046421	20
356260551239284266908724943672911100488558	21
1147374494946449194450825817605340123679150461	22
5032486852040265322461550844695939678052967384053	23
30210545039307528599583618386687349227933725131035504	24
249400383130659050580193267861459579254489822650065685961	25
2844134548699568981561554629043146070324332400944867482340313	26
44993294034522185332489548856700572371349354518671249097245374660	27
991277251392360301443460288397009109066708275778086061470009877027739	28
30526157144572224953157514915475479605501638476250575941226904780179348933	29
1318363800739595427128835554231270770209426196402736248743162258824492158995254	30

5 Conclusion

Our main result is an algorithm that given n, computes the number of labeled chordal graphs on n vertices using $O(n^7)$ arithmetic operations (and in $O(n^9 \log n)$ time in the RAM model). This yields a sampling algorithm that generates a labeled chordal graph on n vertices uniformly at random. For the sampling algorithm, once we have run the counting algorithm as a preprocessing step, each sample can be obtained using $O(n^4)$ arithmetic operations.

The main open problem is to design a substantially faster algorithm for counting or sampling labeled chordal graphs. We presented exact counting and sampling algorithms; nevertheless, allowing for approximate counting/sampling might enable even faster algorithms.

Table 2 Numbers of ω-colorable labeled connected chordal graphs on n vertices. When $\omega = 2$, the algorithm counts labeled trees.

n									
2	3	4	5	6	7	8	9		
1	3	16	125	1296	16807	262144	4782969	2	ω
	4	34	480	9831	268093	9185436	379623492	3	
		35	540	13136	466683	22732032	1437072780	4	
			541	13301	488873	25736782	1873146621	5	
				13302	489286	25863916	1910084529	6	
					489287	25864896	1910751531	7	
						25864897	1910753781	8	
							1910753782	9	

n				
10	11	12		
100000000	2357947691	61917364224	2	ω
18376225525	1019282908941	63707908718994	3	
112588153700	10535042533301	1144261607209084	4	
181962472490	22726623077466	3513611793935959	5	
192919501307	26158547399061	4666697716137194	6	
193325509217	26400465973728	4813890013657154	7	
193328830337	26404655450778	4818876084111431	8	
193328835392	26404671456933	4818917765689886	9	
193328835393	26404671468120	4818917841203841	10	
	26404671468121	4818917841228327	11	
		4818917841228328	12	

To be precise, for approximate sampling we are aiming for an algorithm that, given n and $\delta > 0$, samples from a distribution δ-close to uniform (say in total variation distance) in time polynomial in n and $\log(1/\delta)$, where the dependence on n is significantly less than n^7. Two interesting approaches to consider are Markov Chain Monte Carlo (MCMC) algorithms, such as the chain proposed in [33], and the Boltzmann sampling scheme used in [10] for planar graphs.

Moving beyond chordal graphs, there are many interesting graph classes for which the problem of counting/sampling n-vertex labeled graphs in polynomial time appears to be open, including perfect graphs, weakly chordal graphs, strongly chordal graphs, and chordal bipartite graphs, as well as many graph classes characterized by a finite set of forbidden minors, subgraphs, or induced subgraphs. It is worth noting that for some well-known graph classes of this form, such as planar graphs, polynomial-time algorithms are known [6, 10].

References

1 Andrii Arman, Pu Gao, and Nicholas Wormald. Fast uniform generation of random graphs with given degree sequences. *Random Structures & Algorithms*, 59(3):291–314, 2021.

2 EA Bender, LB Richmond, and NC Wormald. Almost all chordal graphs split. *Journal of the Australian Mathematical Society*, 38(2):214–221, 1985.

3 Ivona Bezáková, Nayantara Bhatnagar, and Eric Vigoda. Sampling binary contingency tables with a greedy start. *Random Structures & Algorithms*, 30(1-2):168–205, 2007.

4 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, 2020.

5 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.

6 Manuel Bodirsky, Clemens Gröpl, and Mihyun Kang. Generating labeled planar graphs uniformly at random. *Theoretical Computer Science*, 379(3):377–386, 2007.

7 Andreas Brandstädt, Van Bang Le, and Jeremy P Spinrad. *Graph classes: a survey*. SIAM, 1999.

8 Tınaz Ekim, Mordechai Shalom, and Oylum Şeker. The complexity of subtree intersection representation of chordal graphs and linear time chordal graph generation. *J. Comb. Optim.*, 41(3):710–735, 2021.

9 Martin Farber. Independent domination in chordal graphs. *Operations Research Letters*, 1(4):134–138, 1982.

10 Éric Fusy. Uniform random sampling of planar graphs in linear time. *Random Structures & Algorithms*, 35(4):464–522, 2009.

11 Pu Gao and Nicholas Wormald. Uniform generation of random regular graphs. *SIAM Journal on Computing*, 46:1395–1427, 2017.

12 Pu Gao and Nicholas Wormald. Uniform generation of random graphs with power-law degree sequences. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1741–1758, 2018.

13 Fănică Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

14 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

15 Catherine Greenhill. *Generating graphs randomly*, pages 133–186. London Mathematical Society Lecture Note Series. Cambridge University Press, 2021.

16 Catherine Greenhill and Matteo Sfragara. The switch Markov chain for sampling irregular graphs and digraphs. *Theoretical Computer Science*, 719:1–20, 2018.

17 Dan Gusfield. The multi-state perfect phylogeny problem with missing and removable data: Solutions via integer-programming and chordal graph theory. In *Research in Computational Molecular Biology (RECOMB)*, pages 236–252, 2009.

18 András Hajnal and János Surányi. Über die auflösung von graphen in vollständige teilgraphen. *Ann. Univ. Sci. Budapest, Eötvös Sect. Math*, 1:113–121, 1958.

19 David Harvey and Joris Van Der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2):563–617, 2021.

20 Pinar Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

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

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

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

24 Asish Mukhopadhyay and Md. Zamilur Rahman. Algorithms for generating strongly chordal graphs. In *Transactions on Computational Science XXXVIII*, pages 54–75, 2021.

25 OEIS Foundation Inc. Entry A058862 in the On-Line Encyclopedia of Integer Sequences, 2023. Published electronically at `http://oeis.org`.

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

27 Heinz Prüfer. Neuer beweis eines satzes über permutationen. *Arch. Math. Phys*, 27(1918):742–744, 1918.

28 Teresa Przytycka, George Davis, Nan Song, and Dannie Durand. Graph theoretical insights into evolution of multidomain proteins. *J Comput Biol.*, 13(2):351–363, 2006.

29 H.N. de Ridder et al. Information system on graph classes and their inclusions (ISGCI). *www.graphclasses.org*, 2016.

30 Donald J Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph theory and computing*, pages 183–217. Elsevier, 1972.

31 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.

32 Oylum Şeker, Pinar Heggernes, Tınaz Ekim, and Z. Caner Taşkın. Linear-time generation of random chordal graphs. In *Proceedings of the 10th International Conference on Algorithms and Complexity (CIAC)*, pages 442–453, 2017.

33 Wenbo Sun and Ivona Bezáková. Sampling random chordal graphs by MCMC (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34(10), pages 13929–13930, 2020.

34 Marcel Wienöbst, Max Bannach, and Maciej Liśkiewicz. Polynomial-time algorithms for counting and sampling Markov equivalent DAGs. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI)*, pages 12198–12206, 2021.

35 Nicholas C. Wormald. Counting labelled chordal graphs. *Graphs and Combinatorics*, 1:193–200, 1985.

36 Nicholas C. Wormald. Generating random unlabelled graphs. *SIAM Journal on Computing*, 16(4):717–727, 1987.

37 Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.