Polynomial Delay Algorithm for Minimal Chordal Completions

Caroline Brosse
Université Clermont Auvergne, Clermont Auvergne INP, CNRS, Mines Saint-Etienne, Limos, F-63000 Clermont-Ferrand, France

Vincent Limouzy
Université Clermont Auvergne, Clermont Auvergne INP, CNRS, Mines Saint-Etienne, Limos, F-63000 Clermont-Ferrand, France

Arnaud Mary
Université de Lyon, Université Lyon 1, CNRS, Laboratoire de Biométrie et Biologie Evolutive UMR 5558, 69622 Villeurbanne, France
ERABLE team, Inria Grenoble Rhône-Alpes, Villeurbanne, France

Abstract

Motivated by the problem of enumerating all tree decompositions of a graph, we consider in this article the problem of listing all the minimal chordal completions of a graph. In [8] (Pods 2017) Carmeli et al. proved that all minimal chordal completions or equivalently all proper tree decompositions of a graph can be listed in incremental polynomial time using exponential space. The total running time of their algorithm is quadratic in the number of solutions and the existence of an algorithm whose complexity depends only linearly on the number of solutions remained open. We close this question by providing a polynomial delay algorithm to solve this problem which, moreover, uses polynomial space.

Our algorithm relies on *Proximity Search*, a framework recently introduced by Conte and Uno [12] (Stoc 2019) which has been shown powerful to obtain polynomial delay algorithms, but generally requires exponential space. In order to obtain a polynomial space algorithm for our problem, we introduce a new general method called *canonical path reconstruction* to design polynomial delay and polynomial space algorithms based on proximity search.

2012 ACM Subject Classification Mathematics of computing → Enumeration; Mathematics of computing → Graph algorithms

Keywords and phrases Graph Algorithm, Algorithmic Enumeration, Minimal chordal completions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.33

Category Track A: Algorithms, Complexity and Games

Related Version *Previous Version*: `https://arxiv.org/abs/2107.05972`

Funding *Caroline Brosse*: ANR project GRALMECO (ANR-21-CE48-0004-01).
Vincent Limouzy: ANR project GRALMECO (ANR-21-CE48-0004-01) and by H2020 Marie Skłodowska-Curie Research and Innovation Staff Exchange European project 691161 "GEO-SAFE".
Arnaud Mary: ANR Project GrR (ANR-18-CE40-0032).

1 Introduction

Since its introduction by Dirac [13], the class of chordal graphs received a lot of attention. The many interesting properties of chordal graphs (*e.g.* a linear number of maximal cliques and minimal separators, a useful intersection model, a specific elimination ordering to cite a few) lead to the design of efficient algorithms for problems that are usually difficult on general graphs. On top of that, chordal graphs are closely connected to an important graph parameter called *treewidth* and its associated *tree-decomposition* introduced independently by Halin [14] and Robertson and Seymour [22].


Treewidth has played an important role in algorithmics for the last forty years, since its introduction and popularization by Robertson and Seymour. This popularity is deserved: given a tree decomposition of small width, many problems that are usually hard become solvable in polynomial time.

From the structure of maximal cliques known for chordal graphs, it is well-known that an optimal tree-decomposition can be computed in linear time on this class. For general graphs, one way to define the treewidth is to find the smallest value k such that the graph is a subgraph of a k-tree, that is to say, a subgraph of a chordal graph with maximum clique size at most $k+1$. Unfortunately, it was shown by Arnborg, Corneil & Proskurowski [1] that determining whether a graph is a partial k-tree (*i.e.* a subgraph of a k-tree) is NP-complete. In a different direction, the *minimum fill-in* problem asks to find the minimum number of edges to add to the graph in order to turn it into a chordal graph. Yannakakis [27] proved that the minimum fill-in problem is NP-hard.

Since the aforementioned problems are intractable, relaxations of these problems have been considered. The problem of computing chordal completion that are *inclusion-wise minimal*, has been intensively studied, either to find an optimal tree decomposition with an exponential-time algorithm or to find one tree decomposition in polynomial-time. Over the past decades, numerous polynomial algorithms have been provided to compute one minimal chordal completion [3, 5, 24, 15, 19, 20, 23].

However, from an application point of view, computing only one tree decomposition might not be satisfactory. For that reason, Carmeli et al. [7, 8] considered the problem of listing all the minimal chordal completions of a graph, and hence obtaining all the minimal tree decompositions of a graph. In the same line of research, Ravid et al. [21] considered the same problem by adding a requirement on the order in which solutions are produced.

An enumeration algorithm lists every solution of a given problem exactly once. Since the considered problem can have exponentially (in the input size) many solutions, the traditional complexity measures are no longer relevant. Instead, the common approach called *output sensitive* analysis is to bound the time complexity by a function of the input and the output size. Johnson et al. adapted in [17] the notion of polynomial time algorithm for enumeration algorithms. An algorithm is said to be *output polynomial* if its complexity can be bounded by a polynomial function expressed in the size of both the input and the output. As the number of solutions might be huge, this notion is not fine enough to capture the efficiency of the algorithm. For that reason, Johnson et al. further refined this notion by introducing the *incremental polynomial time*, meaning that the time used to produce a new solution of the problem is bounded by a polynomial in the size of the input and the size of all the already produced solutions. They strengthened the concept with the notion of *polynomial delay*: the time between the generation of two consecutive solutions is bounded by a polynomial in the size of the input only. The total running time of a polynomial delay algorithm depends only linearly on the size of the output, and since all solutions have to be outputted, this dependency is optimal.

For the enumeration of minimal triangulations of a graph, Carmeli et al. in [7, 8] presented a highly non-trivial algorithm. Their algorithm runs in *incremental polynomial time*; its total complexity is quadratic in the number of solutions, and to avoid duplication of solutions, requires exponential space. The algorithm is based on a result by Parra & Scheffler [20], according to which there is a bijection between the minimal triangulations of a graph and the maximal independent sets of a special graph of minimal separators of the input graph. In [8], they proved under *Exponential Time Hypothesis* that their approach cannot be improved to achieve polynomial delay. This intractability result also holds for the exponential space.

In [7, 8] and at a Dagstuhl seminar [4], it was left as an open problem whether a polynomial delay and/or a polynomial space algorithm for this problem could be obtained. We answer this question by the affirmative. Our main result is the following theorem.

▶ **Theorem 1.** *The minimal chordal completions of a graph can be computed in polynomial delay and polynomial space.*

In addition, our algorithm is simple and can be easily implemented. Since it is a polynomial delay algorithm, its total complexity is linear in the number of solutions. It contrasts with the quadratic one of [7, 8].

The paper is organised as follows. In Section 2 we recall the results and techniques used in algorithmic enumeration. In Section 3, we present the concepts that will be used for minimal chordal completions, and we present a first polynomial delay and exponential space algorithm. Then in Section 4 we present our main result, namely a polynomial delay and polynomial space algorithm to list all the minimal chordal completions of a graph without duplication. Finally, in Section 5 we formalize the framework used in Section 4 by introducing a new general method called *canonical path reconstruction* to design polynomial space enumeration algorithms.

2 Definitions and preliminary results

Throughout the article, standard graph theory notations will be used. A graph, always assumed to be simple, finite and undirected, is denoted by $G = (V, E)$ where V is the set of vertices and E is the set of edges. When the context is ambiguous, notations $V(G)$ and $E(G)$ can be used. For any $k \geq 3$, C_k denotes a cycle of length k.

We denote by E^c the set of *non-edges* of G, that is, the complement of the set E in the larger set of all two-element subsets of V. Then, for a subset $F \subset E^c$ we denote by $\bar{F}$ the set $E^c \setminus F$.

A graph is *chordal*, or *triangulated*, if it does not contain any induced cycle of length 4 or more. In other words, every cycle of length more than 3 of a chordal graph has at least one chord (*i.e.* an edge that connects non-consecutive vertices of the cyle).

Given a graph G and a supergraph H of G on the same vertex set, H is called a *triangulation* of G if H is chordal, and the edges of H which are not edges of G are called *fill edges*. In the whole paper, triangulations, or chordal completions, will be identified with the set of fill edges they induce. This is why for a graph $G = (V, E)$, we call a set $F \subseteq E^c$ a *chordal completion of* G if the supergraph $G_F = (V, E \cup F)$ is chordal. Let us denote by $\mathcal{F}$ the set of all chordal completions of G and by $\mathcal{F}_{\min}$ the set of minimal chordal completions of G, with respect to inclusion. A characterisation of minimal chordal completions is given in [23, Theorem 2]: a chordal completion F of a graph G is minimal *if and only if* for any $f \in F$, the graph $G_F \setminus \{f\}$ has an induced C_4.

From now on, $G = (V, E)$ is considered to be an arbitrary input graph that has no particular property and is therefore not assumed to be chordal. In the whole article, notation n is used for the number of vertices of G, that is, $n = |V|$. Finally, as our goal is to enumerate all minimal chordal completions of G, we may simply refer to them as "minimal completions".

The enumeration of minimal chordal completion takes place in the more general task of enumerating the minimal or the maximal subsets of a set system. A *set system* is a couple $(\mathcal{U}, \mathcal{F})$ where $\mathcal{U}$ is called the ground set and $\mathcal{F} \subseteq 2^{\mathcal{U}}$ is a family of subsets of $\mathcal{U}$. For a set system $(\mathcal{U}, \mathcal{F})$ we denote by $\mathcal{F}_{\min}$ (resp. $\mathcal{F}_{\max}$) the inclusion-wise minimal (resp. maximal) sets in $\mathcal{F}$. Many enumeration problems consist in enumerating the set $\mathcal{F}_{\max}$ or $\mathcal{F}_{\min}$ of a set

system $(\mathcal{U}, \mathcal{F})$. Of course the set $\mathcal{F}$ is usually not part of the input and we simply assume that a polynomially computable oracle membership is given, *i.e.* one can check whether a subset $F \subseteq \mathcal{U}$ belongs to $\mathcal{F}$ in time polynomial in $\mathcal{U}$.

In [12], the authors describe a method called *Proximity Search* (by canonical reconstruction) to design polynomial delay algorithms to enumerate $\mathcal{F}_{\max}$ of a set system $(\mathcal{U}, \mathcal{F})$. Given a set family $\mathcal{F}$ on a ground set $\mathcal{U}$, an *ordering scheme* π is a function which associates for every $F \in \mathcal{F}$ a permutation $\pi(F) = f_1, ..., f_{|F|}$ of the elements of F such that for all $i < |F|$, the ith prefix $\{f_1, ..., f_i\}$ of $\pi(F)$ is in $\mathcal{F}$. Notice that a set system $(\mathcal{U}, \mathcal{F})$ has an ordering scheme if and only if for every $F \in \mathcal{F}$, there exists $f \in F$ such that $F \setminus \{f\} \in \mathcal{F}$. Set systems having this property are called *accessible*.

The method described in [12] is based on the *proximity* between 2 solutions. While this notion has been defined in a very general context, most of its use cases are based on an ordering scheme. Given an ordering scheme π, and given $F_1, F_2 \in \mathcal{F}$ with $\pi(F_2) = f_1, ..., f_{|F_2|}$, the *$\pi$-proximity* between F_1 and F_2, denoted by $F_1 \tilde{\cap} F_2$, is the largest $i \leq |F_2|$ such that $\{f_1, ..., f_i\} \subseteq F_1$. It is worth noticing that the proximity relation between two solutions is not necessarily symmetric.

A polynomial-time computable function $\textsc{Neighbours} : \mathcal{F}_{\max} \to 2^{\mathcal{F}_{\max}}$ is called *π-proximity searchable*, if for every $F_1, F_2 \in \mathcal{F}_{\max}$ there exists $F' \in \textsc{Neighbours}(F_1)$ such that $F' \tilde{\cap} F_2 > F_1 \tilde{\cap} F_2$. Finally, we say by extension that an ordering scheme π of a set system $(\mathcal{U}, \mathcal{F})$ is *proximity searchable* if there exists a π-proximity searchable function Neighbours.

One of the major results of [12] is the following theorem.

▶ **Theorem 2** ([12]). *Let $(\mathcal{U}, \mathcal{F})$ be a set system and assume that one can find in polynomial time a maximal set $F_0 \in \mathcal{F}_{max}$. If $\mathcal{F}$ has a proximity searchable ordering scheme, then $\mathcal{F}_{max}$ can be enumerated with polynomial delay.*

The proof of this theorem is based on the analysis of the *supergraph of solutions*. The supergraph of solutions is the directed graph having $\mathcal{F}_{\max}$ as vertex set and there is an arc from F_1 to F_2 if $F_2 \in \textsc{Neighbours}(F_1)$ (where Neighbours is the π-proximity searchable function). The proximity searchability of the ordering scheme implies that this supergraph of solutions is strongly connected. Then, the polynomial delay algorithm consists in starting from an arbitrary solution $F_0 \in \mathcal{F}_{\max}$ and performing a traversal of the supergraph of solutions, following at each step the arcs computed on the fly by Function Neighbours. The strong connectivity of the supergraph of solutions ensures that all solutions will be found. However, with this method, one needs to store in memory the solutions already visited, which in general results in the need of an exponential space.

One of the classical methods in enumeration to avoid the storage of the already outputted solutions is to define a parent-child relation over the set of solutions. It consists in associating to each $F \in \mathcal{F}_{\max} \setminus F_0$ a parent solution $\textsc{Parent}(F) \in \mathcal{F}_{\max}$ such that $F \in \textsc{Neighbours}(\textsc{Parent}(F))$. Given the Parent function, one can finally define the Children as $\textsc{Children}(F) := \{F' \in \textsc{Neighbours}(F) \mid \textsc{Parent}(F') = F\}$. The goal is to define the function Parent in such a way that the arcs of the supergraph of solutions defined by Function Children form a spanning arborescence of the supergraph of solutions, because in such a situation, one does not need to store the already visited solutions in a traversal of the supergraph of solutions. This method and the way to traverse the spanning arborescence of the supergraph is called *Reverse Search* [2] and has been used in many contexts.

Most applications of Reverse Search use a classical parent-child relation originally introduced in [25] for the enumeration of maximal independent sets of a graph. This specific parent-child relation has been used in general enumeration frameworks to obtain polynomial delay and polynomial space algorithms [18, 9]. Unfortunately, this method can only be used for hereditary set systems and it is not compatible with Proximity Search in general.

In [10] the authors show how to adapt this parent-child relation to commutable set systems (a class that strictly contains hereditary set systems) and in [11], it has been shown that the same method can be combined with Proximity Search to obtain polynomial delay and polynomial space algorithms for commutable set systems.

A set system is *commutable* if for any $X, Y \in \mathcal{F}$ with $X \subseteq Y$, the following two conditions hold:

- **Strong accessibility:**
  There exists $f \in Y \setminus X$ such that $X \cup \{f\} \in \mathcal{F}$
- **Commutability:**
  For any $a, b \in Y \setminus X$, if $X \cup \{a\} \in \mathcal{F}$ and $X \cup \{b\} \in \mathcal{F}$ then $X \cup \{a, b\} \in \mathcal{F}$

More formally, the authors introduce the notion of *prefix-closed* ordering schemes (*cf.* Section 5 for a definition) and they show the following theorem.

▶ **Theorem 3** ([11]). *Let $(\mathcal{U}, \mathcal{F})$ be a commutable set system and assume that one can find in polynomial time a maximal set $F_0 \in \mathcal{F}_{max}$. If $\mathcal{F}$ has a proximity searchable prefix-closed ordering scheme, then $\mathcal{F}_{max}$ can be enumerated with polynomial delay and polynomial space.*

The polynomial space complexity comes from the definition of a parent-child relation that strongly relies on the commutability property. In the current paper, we introduce a new way of defining parent-child relations called *canonical path reconstruction* that does not require the commutability property. Since the set of chordal completions is not a commutable set system, this new method will be used in Section 4 to obtain a polynomial delay algorithm. Then, in Section 5 we improve Theorem 3 by proving the following which applies to non-commutable set systems.

▶ **Theorem 4.** *Let $(\mathcal{U}, \mathcal{F})$ be a set system and assume that one can find in polynomial time a maximal set $F_0 \in \mathcal{F}_{max}$. If $\mathcal{F}$ has a proximity searchable prefix-closed ordering scheme, then $\mathcal{F}_{max}$ can be enumerated with polynomial delay and polynomial space.*

3 Enumeration of minimal chordal completions: polynomial delay

In this section we present a polynomial delay and exponential space algorithm to list all minimal chordal completions of a graph. In Section 3.1 we present all the concepts necessary to define an ordering scheme that will suit to minimal chordal completions. Then in Section 3.2, we present the neighbouring function and prove that this function is proximity searchable. As a consequence, together with Theorem 2, we obtain a polynomial delay algorithm.

3.1 Ordering scheme for chordal graphs

A class $\mathcal{C}$ of graphs is called *sandwich-monotone* [16] if for any two graphs H_1 and H_2 in $\mathcal{C}$ such that $E(H_1) \subsetneq E(H_2)$, there exists an edge $e \in E(H_2) \setminus E(H_1)$ such that $H_2 \setminus \{e\}$ is also in $\mathcal{C}$. This property is equivalent to being *strongly accessible* in terms of set systems (see for example [10]).

In [23, Lemma 2], the authors proved that the class of chordal graphs is sandwich-monotone. This immediately implies that if $F_1 \subsetneq F_2$ are two chordal completions of any graph, there exists $e \in F_2 \setminus F_1$ such that $F_2 \setminus \{e\}$ is a chordal completion, or equivalently there exists $e \in F_2 \setminus F_1$ such that $F_1 \cup \{e\}$ is a chordal completion. That rephrases as the following lemma.

▶ **Lemma 5.** *Chordal graphs are sandwich-monotone. In particular, chordal completions of G are sandwich-monotone.*

The general idea is then to use the sandwich-monotonicity of chordal completions to derive a suitable ordering scheme. The Proximity Search framework introduced in [10] has been designed to enumerate inclusion-wise maximal subsets of a set system. However, the same framework could be applied to the enumeration of inclusion-wise minimal subsets of a set system, by considering the complements of the solutions. To this effect, we will not work directly with the edge sets of the completions but rather with the sets of their *non-edges*. That is to say, we will consider for any completion F its complement $\bar{F} := E^c \setminus F$. The goal of this section is then to apply Proximity Search to the set $\bar{\mathcal{F}} := \{\bar{F} \mid F \in \mathcal{F}\}$ in order to enumerate the set $\bar{\mathcal{F}}_{\max} := \max(\bar{\mathcal{F}}) = \{\bar{F} \mid F \in \mathcal{F}_{min}\}$.

From now on, the elements of E^c (that is to say, the non-edges of G) are assumed to be arbitrarily ordered. The following definitions are similar to the ones given in [10] but are defined on $\mathcal{F}$ instead of $\bar{\mathcal{F}}$. Let F be a chordal completion of G and X be any subset of E^c. Consider the set X as the set from which we are allowed to remove elements. We define:

- $\textsc{Candidates}(F, X) := \{e \in X \cap F : F \setminus \{e\} \in \mathcal{F}\}$
- $\textsc{Candidates}(F) := \textsc{Candidates}(F, F)$
- $c(F, X) := \min(\textsc{Candidates}(F, X))$
- $c(F) := \min(\textsc{Candidates}(F))$

Now, given a chordal completion F and a set $X \subseteq E^c$, we denote by $\textsc{Del}(F, X)$ the chordal completion included in F by iteratively removing $c(F, X)$ from F at each step. Finally, we define $\textsc{Del}(F) := \textsc{Del}(F, F)$. Note that, for any F, X, computing $\textsc{Del}(F, X)$ corresponds to the following procedure.

Function Del(F, X).

while $\textsc{Candidates}(F, X) \neq \emptyset$ **do**
  remove $c(F, X)$ from F;
return F

▶ **Remark 6.** *By Lemma 5, if $F \in \mathcal{F}$, then $\textsc{Del}(F) \in \mathcal{F}_{min}$. That is to say, Del can be used to turn a chordal completion into a minimal one in a canonical way.*

Also, as E^c is a chordal completion (it corresponds to the clique completion) of G, the next lemma holds.

▶ **Lemma 7.** *If F is a minimal chordal completion of G, then $\textsc{Del}(E^c, \bar{F}) = F$.*

Proof. By Remark 6, it holds $\textsc{Del}(E^c, \bar{F}) \in \mathcal{F}_{\min}$, and $F \subset \textsc{Del}(E^c, \bar{F})$. As a result, since the only minimal solution containing F is F itself, then $\textsc{Del}(E^c, \bar{F}) = F$. ◀

The procedure followed to compute $\textsc{Del}(E^c, \bar{F})$ provides an ordering on the elements of $\bar{F}$ by considering the order in which the elements of $\bar{F}$ are removed to obtain F. From this ordering, we can define for any chordal completion F the canonical ordering $\textsc{Can}(\bar{F}) := s_1, ..., s_{|\bar{F}|}$ of $\bar{F}$ as follows:

- $s_1 := c(E^c, \bar{F})$;
- for all $1 \leq i < |\bar{F}|$, $s_{i+1} := c(E^c \setminus \{s_1, ..., s_i\}, \bar{F})$.

For a set $\bar{F}$ and its canonical ordering $\textsc{Can}(\bar{F}) := s_1, ..., s_{|\bar{F}|}$ we define $\bar{F}^i$ as the set of elements $\{s_1, ..., s_i\}$ of $\bar{F}$ and we define $F^i := E^c \setminus \bar{F}^i$. By definition of $\textsc{Candidates}(E^c, \bar{F})$, any prefix $\bar{F}^i$ of this ordering belongs to $\bar{\mathcal{F}}$. In other words, F^i denotes the chordal completion of G, not necessarily minimal, obtained by removing the ith prefix of $\textsc{Can}(\bar{F})$ from the clique completion. Thus, the canonical ordering Can is an ordering scheme of $\bar{\mathcal{F}}$.

This ordering will be used to measure the proximity between two solutions in order to call the Proximity Search algorithm for minimal chordal completions. Note that it can be computed in polynomial time for any solution, the complexity being essentially this of calling Function Del.

We now define the notion of proximity between two solutions that will be used in the sequel. For F_1 and F_2 two minimal completions of G, let $\textsc{Can}(\bar{F_2}) = f_1, ..., f_k$ be the canonical ordering of $\bar{F_2}$. The *proximity* $\bar{F_1} \tilde{\cap} \bar{F_2}$ between $\bar{F_1}$ and $\bar{F_2}$ is defined as the largest $i \leq k$ such that $\{f_1, ..., f_i\} \subseteq \bar{F_1}$. It is worth noticing that the proximity relation between two solutions is not necessarily symmetric. The notion defined here corresponds to the "standard" proximity measure that is usually adopted when using the Proximity Search framework [12, 10, 6]. Note that the proximity is defined here on the set of elements of E^c (that is to say, non-edges of G) which do *not* belong to the chordal completion. Since we are working on both the completions and their complement, by a slight abuse of language, when we speak about the proximity between two minimal completions, we actually mean the proximity between their complement sets.

3.2 Polynomial delay algorithm

We show in this section that the ordering scheme Can defined above is proximity searchable. Since the idea is to consider the complement sets of minimal completions rather than the completions themselves, we will seek to maximise the set of common *non-edges* between two minimal completions in order to increase the proximity. So, we actually show that Can is a proximity searchable ordering scheme of $\bar{\mathcal{F}}$.

The first goal is to define a suitable neighbouring function on the class of chordal graphs. Any solution must have a polynomial number of neighbours, each of them being computable in polynomial time in order to guarantee the polynomial delay when applying Proximity Search.

Given a chordal graph H and an edge $e = \{x, y\} \in E(H)$, the *flip operation* $\textsc{Flip}(H, e)$ consists in removing e from H, and turning the common neighbourhood of x and y into a clique. More formally, if $H' = \textsc{Flip}(H, e)$, then $V(H') = V(H)$ and $E(H') := (E(H) \setminus \{e\}) \cup \{uv \mid u, v \in N_H(x) \cap N_H(y)\}$. The flip operation is illustrated in Figure 1.

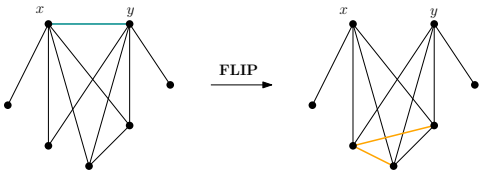

Figure 1 The flip operation on $e = \{x, y\}$. The common neighbourhood of x and y is turned into a clique.

Since H is chordal, the removal of e can create several chordless C_4s of which e was the only chord. We will see that completing the common neighbourhood of x and y into a clique adds the missing chords to all these C_4. In other words, the flip operation preserves the class of chordal graphs, as stated next.

▶ **Lemma 8.** *Let H be a chordal graph, and $e = \{x, y\}$ be an edge of H. Then the graph $H' := \textsc{Flip}(H, e)$ is also chordal.*

Proof. Assume (for contradiction) that H' contains a chordless cycle C of length $\ell > 3$.

Suppose C contains only edges of H. Since H is chordal, it does not contain long induced cycles. Then necessarily C is a C_4 of H of which e was the unique chord, otherwise there would be an induced cycle of length at least 4 (either C or a cycle made from e and edges of C) in the chordal graph H. By definition, the flip operation adds an edge between the other two vertices of C, meaning that C has a chord in H'. This cannot happen.

Therefore, C contains an edge $e' = \{z, t\}$ added by the flip operation (*i.e.* e' is an edge of H' but not of H). Since C contains the edge e', it does not contain any other fill edge added by the flip operation, otherwise a chord of C would also be added by the flip. Plus, we assumed that e' is added by the flip operation, so necessarily both z and t belong to $N_H(x) \cap N_H(y)$. Therefore, $P := C - \{e'\}$ is a $z - t$ path of H', disjoint from x and y. Remark that P is an induced path of H since it contains only edges of H, that is, z and t are in the same connected component of $H \setminus \{x, y\}$.

▷ Claim 9. Every vertex of P is in $N_H(x) \cap N_H(y)$.

Proof. Suppose that there exists a vertex s of P which is not a neighbour of x. We will show that there exists in H a chordless cycle containing s and x.

As s is not a neighbour of x, and z is by hypothesis, there exists $q \in N(x)$ a vertex of P that is between z and s, and closest to s on P. Similarly, let $r \in N(x)$ be a vertex of P that is between t and s, and closest to s.

The $q - r$ subpath of P induced by all vertices between q and r is therefore a chordless path of H of which all internal nodes are non-neighbours of x. It follows that adding x to this $q - r$ subpath creates a chordless cycle of length at least 4 in H. This is excluded, so all vertices of P are neighbours of x in H.

By symmetry in x and y, we also deduce that all vertices of P are neighbours of y.

The claim is proved: every vertex of P is in $N_H(x) \cap N_H(y)$. ◁

By the previous Claim, the flip operation turns the cycle C into a clique. Hence C has length 3, a contradiction. ◀

When dealing with chordal completions, the notation of the flip operation will be slightly adapted: for $F \in \mathcal{F}$, and $e \in F$ (e is always chosen among the fill edges of the completion), we write $\textsc{Flip}(F, e)$ instead of $\textsc{Flip}(G_F, e)$.

From Lemma 8, we are then able to deduce the following.

▶ **Lemma 10.** *Let F be a chordal completion of a graph G. If $F \in \mathcal{F}_{min}$, and $e \in F$, then $\textsc{Flip}(F, e) \in \mathcal{F}$.*

Proof. The only edge that is in F but not in $\textsc{Flip}(F, e)$ is e. Since $e \in F$, it implies that all edges of G are still in $\textsc{Flip}(F, e)$. As $\textsc{Flip}(F, e)$ is chordal by Lemma 8. ◀

Observe that $\textsc{Flip}(F, e)$ is a chordal completion of G that is not necessarily minimal.

We are now ready to explain the neighbouring function used to enumerate $\mathcal{F}_{\min}$. Given $F \in \mathcal{F}_{\min}$ and $e \in F$, we define the successor of F according to e as the minimal completion $\textsc{Succ}(F, e) := \textsc{Del}(\textsc{Flip}(F, e))$. We now define the neighbours of a solution F as $\textsc{Neighbours}(F) := \{\textsc{Succ}(F, f) | f \in F\}$. As a corollary of Remark 6 and Lemma 10, it is true that $\textsc{Succ}(F, e) \in \mathcal{F}_{\min}$. Observe also that each solution has a polynomial number of neighbours since $\textsc{Neighbours}(F) \leq |F|$.

Now the neighbouring function is properly defined, it is easy to notice that it can be computed in polynomial time. It remains to prove that the function NEIGHBOURS is CAN-proximity searchable. This will allow us to use Proximity Search on the set of minimal chordal completions of a graph G.

▶ **Lemma 11.** *Let F_1 and F_2 be two minimal chordal completions of a graph G. Let $f_1, \dots, f_k = \text{CAN}(\bar{F}_2)$ and let $i := \bar{F}_1 \tilde{\cap} \bar{F}_2$. Then the following statements hold:*
1. *$f_{i+1} \notin \bar{F}_1$;*
2. *$\overline{\text{SUCC}(F_1, f_{i+1})} \tilde{\cap} \bar{F}_2 > i$.*

Proof. 1. By definition of i as the length of the largest prefix of $\text{CAN}(\bar{F}_2)$ included in $\bar{F}_1$, it holds $f_{i+1} \notin \bar{F}_1$.

2. Let $F' := \text{FLIP}(F_1, f_{i+1})$, we will show that $\{f_1, ..., f_{i+1}\} \subseteq \bar{F'}$. Since $\text{SUCC}(F_1, f_{i+1}) \subseteq F'$, it will imply that $\{f_1, ..., f_{i+1}\} \subseteq \overline{\text{SUCC}(F_1, f_{i+1})}$, and finally $\overline{\text{SUCC}(F_1, f_{i+1})} \tilde{\cap} \bar{F}_2 \geq i+1$.

First, notice that by definition of FLIP, $f_{i+1} \notin F'$. Let x and y be the two endpoints of f_{i+1}. Assume for contradiction that an edge f_j, $j \leq i$ is added when completing $N_{G_{F_1}}(x) \cap N_{G_{F_1}}(y)$ into a clique. As stated earlier, the notation $\bar{F}_2^{\,i+1}$ is used to denote the set $\{f_1, ..., f_{i+1}\}$, which is included in $\bar{F}_2$, and F_2^{i+1} represents the associated chordal completion.

Let u and v be the two endpoints of f_j. Then x, u, y, v form a C_4 of which f_{i+1} was the unique chord in G_{F_1}. By definition of $\text{CAN}(\bar{F}_2)$, F_2^{i+1} is a chordal completion of G, and since neither the chords f_j nor f_{i+1} belong to it, at least one of the edges xu, uy, yv or vx does not belong to F_2^{i+1} since otherwise x, u, y, v would form a chordless C_4 in F_2^{i+1}. Assume without loss of generality that $xu \notin F_2^{i+1}$. Since $\bar{F}_2^{\,i+1} = \{f_1, ..., f_{i+1}\}$, there exists $\ell \leq i$ such that $xu = f_\ell$. But then we have found an edge $f_\ell \notin \bar{F}_1$ with $\ell \leq i$, which contradicts the assumption $\bar{F}_1 \tilde{\cap} \bar{F}_2 = i$.

Consequently, $\overline{\text{SUCC}(F_1, f_{i+1})} \tilde{\cap} \bar{F}_2 > i$. ◀

As a consequence, since the polynomial computable function SUCC is able to increase the proximity between 2 solutions, it proves that the ordering scheme defined by CAN is proximity searchable. More formally, the function $\text{NEIGHBOURS}(\bar{F}) := \{\bar{X} \mid X \in \text{NEIGHBOURS}(F)\}$ is a CAN-proximity searchable function and CAN is then a proximity searchable ordering scheme for $\bar{\mathcal{F}}_{max}$.

Therefore, by Theorem 2, $\bar{\mathcal{F}}_{max}$ or equivalently $\mathcal{F}_{\min}$ can be enumerated with polynomial delay. This is summarized in the following theorem.

▶ **Theorem 12.** *There exists a polynomial delay algorithm for the enumeration of minimal chordal completions of a graph.*

To prove that CAN is proximity searchable, we only used the fact that it is an ordering scheme of $\bar{\mathcal{F}}$. Thus any ordering scheme of $\bar{\mathcal{F}}$ is actually proximity searchable.

Yet, applying Proximity Search usually gives polynomial delay with exponential space: to avoid duplication it is necessary to store all the generated solutions in a lookup table. Each newly generated solution is then searched in the table in polynomial time.

4 Polynomial space

We would like to make the enumeration process work in polynomial space, since this could lead to an algorithm that is more usable in practice. As stated in Theorem 3, it is proven in [10] that if the ordering scheme is prefix-closed and $\mathcal{F}$ is a *commutable* set system, one can design a polynomial delay and polynomial space algorithm for the enumeration of $\mathcal{F}_{\max}$.

An ordering scheme π over $\mathcal{F}$ is called *prefix-closed*, if for all $F \in \mathcal{F}$, there exists an ordering $<_F$ of the elements of $\textsc{Candidates}(F)$ such that $\pi(\bar{F}) = f_1, ..., f_\ell$ *if and only if* for any $i < \ell$, it holds $f_{i+1} = \min_{<_{F^i}}(\textsc{Candidates}(F^i) \cap \bar{F})$. We know by definition of $\textsc{Can}$ that this ordering is prefix-closed (with the ordering of E^c). By Section 3, $\textsc{Can}$ is also proximity searchable. As stated earlier, chordal completions form a strongly accessible set system since they are sandwich-monotone, and so are their complements. However, the set of chordal completions is not a commutable set system, and neither is the set of their complements.

This section is devoted to the definition of a suitable parent-child relation over $\mathcal{F}_{\min}$ which does not require the system to be commutable, in order to obtain a polynomial space algorithm. In the supergraph of solutions, we identify a reference solution named F_0 and we manage to relate any other solution F to F_0. To uniquely determine the parent of a solution, we introduce a new concept called *canonical path reconstruction*. The idea is, for any solution F distinct from F_0, to identify a path in the supergraph of solutions from F_0 to F in a unique manner. Then the parent of solution F is defined as the immediate predecessor of F on this path. In addition, we are able to compute this path in polynomial time for any solution.

To ensure that such a path can be uniquely determined and computed in polynomial time we rely on the specific structure of chordal completions and more specifically on the fact that the canonical ordering we defined on chordal completions is prefix-closed. To the best of our knowledge, it is the first algorithm to define a parent-child relation in this manner.

The parent-child relation will define a spanning arborescence of the supergraph of solutions rooted at F_0. This way, the enumeration algorithm sums up in performing a traversal of the tree in a depth-first manner.

It is somehow counter-intuitive to observe that the canonical path of a solution is not necessarily part of the final arborescence rooted at F_0. This canonical path is only computed to find the last solution in the path before F to define it as the parent of F. We will prove that while the so defined parent-child relation does not exactly follow the canonical paths, it still forms a spanning arborescence of the solution set rooted at F_0.

Let us consider the total ordering $\prec$ over $\bar{\mathcal{F}}_{min}$ defined as $\bar{F}_1 \prec \bar{F}_2$ if $\textsc{Can}(\bar{F}_1)$ is lexicographically smaller than $\textsc{Can}(\bar{F}_2)$.

▶ **Lemma 13.** *Let $F_1, F_2 \in \mathcal{F}_{min}$ with $\textsc{Can}(\bar{F}_1) = f_1, ..., f_\ell$ and $\textsc{Can}(\bar{F}_2) = t_1, ..., t_k$. Assume furthermore that there exists $j \leq \ell$ such that $\{f_1, ..., f_j\} \subset \bar{F}_2$. Then one of the two possibilities holds:*

1. *$\bar{F}_2 \prec \bar{F}_1$;*
2. *$f_i = t_i$ for all $i \leq j$.*

Proof. Let i^* be the smallest index such that $t_{i^*} \neq f_{i^*}$. If $i^* > j$, then 2 holds. Else, $i^* \leq j$. In this case, we deduce from the minimality of i^* that

$$\bar{F_1}^{i^*-1} = \{f_1, ..., f_{i^*-1}\} = \{t_1, ..., t_{i^*-1}\} = \bar{F_2}^{i^*-1} .$$

Since $f_{i^*} \in \bar{F}_2$ and since $f_{i^*} \in \textsc{Candidates}(F_1^{i^*-1}, \bar{F}_1)$ by definition of $\textsc{Can}(\bar{F}_1)$, we deduce $f_{i^*} \in \textsc{Candidates}(F_2^{i^*-1}, \bar{F}_2)$. The canonical ordering $\textsc{Can}$ is prefix-closed, therefore we know that $t_{i^*} = \min(\textsc{Candidates}(F_2^{i^*-1}, \bar{F}_2))$. We deduce from it that $t_{i^*} \leq f_{i^*}$ and since $t_{i^*} \neq f_{i^*}$, we have $t_{i^*} < f_{i^*}$ which proves $\bar{F}_2 \prec \bar{F}_1$. ◀

The algorithm starts by computing $F_0 := \textsc{Del}(E^c)$ in polynomial time. This solution F_0 will be used as the reference solution. Note that we could start from any solution and the results would still be valid, but for simplicity we start from a solution that can easily be identified.

For a minimal chordal completion F with canonical ordering $\textsc{Can}(\bar{F}) := f_1, ..., f_\ell$, we define the *canonical path of* F as the sequence of minimal completions $F_0, ..., F_k = F$ starting at F_0 such that for all $1 \leq i \leq k$, $F_i = \textsc{Succ}(F_{i-1}, f_{(\bar{F}_{i-1} \tilde{\cap} \bar{F})+1})$. Remark that the edge $f_{(\bar{F}_{i-1} \tilde{\cap} \bar{F})+1}$ is part of F_{i-1} and the call to Succ is correct. Let us also note that by construction, the proximity strictly increases along the path. Then, $\textsc{Parent}(F)$ is defined for any $F \neq F_0$ as the last minimal completion F_{k-1} in the canonical path of F.

As remarked before, except F_{k-1}, the other ancestors of F may not be part of its canonical path. For instance F_{k-2} may not be the parent of F_{k-1}.

The detailled Parent function is presented 11.

Function Parent(F).

Compute $f_1, \ldots, f_\ell := \textsc{Can}(\bar{F})$;
Compute $F_{current} := \textsc{Del}(E^c)$;
while $F_{current} \neq F$ **do**
  $f := f_{(\overline{F_{current}} \tilde{\cap} \bar{F})+1}$;
  $F_{prec} := F_{current}$;
  $F_{current} := \textsc{Succ}(F_{prec}, f)$;
return F_{prec}

Then we define $\textsc{Children}(F) := \{F' \in \textsc{Neighbours}(F) \mid \textsc{Parent}(F') = F\} = \{\textsc{Succ}(F, e) \mid e \in F,\ \textsc{Parent}(\textsc{Succ}(F, e)) = F\}$.

▶ **Lemma 14.** *Let F be a minimal chordal completion of G, and let $F_0, \ldots, F_k = F$ be the canonical path of F. For all $j \leq k$, $\bar{F}_j \tilde{\cap} \bar{F} \geq j$.*

Proof. By construction of the canonical path of F, the proximity strictly increases at each step by at least one. ◀

By Lemma 14 the length of the canonical path of F is smaller than $|\bar{F}|$. Since the construction of the canonical path of F is done by applying at most $|\bar{F}|$ times Function Succ, $\textsc{Parent}(F)$ is computable in polynomial time for any $F \in \mathcal{F}_{\min}$.

Now, since the computation of $\textsc{Children}(F)$ is done by applying Function Succ followed by Function Parent at most $|F|$ times, it is also computable in polynomial time.

Once the Children function is defined, one just has to apply the classical algorithm to visit and output all solutions whose high-level description is presented in Algorithm 1. To prove the correctness of the algorithm, we just have to prove that the function Children defines a spanning arborescence on the set of solutions rooted at F_0.

Observe that the naive implementation of Algorithm 1 may use exponential space, since the height of the recursion tree might be exponential. Indeed the recursion tree corresponds to the arborescence defined by the parent-child relation and we are not able to guarantee that its height is polynomial. However, since the functions Parent and Children are computable in polynomial time we don't need to store the state of each recursion call. The classical trick introduced in [25] and formalized in [2] as part of the Reverse Search algorithm is to use Function Parent to perform the backtrack operation on the fly. Indeed this function is able to navigate backward in the tree, removing the need of keeping in memory all recursion calls. Thus, the algorithm only needs to keep in memory the current solution.

▶ **Theorem 15.** *Algorithm 1 outputs all minimal chordal completions of the input graph without duplication, with polynomial delay and using polynomial space.*

Algorithm 1 Efficient enumeration of minimal chordal completions.

input : A graph $G = (V, E)$
output : All minimal chordal completions of G

$F_0 := \textsc{Del}(E^c)$;
Call $\textsc{enum}(F_0)$;

Function `enum` *(F)*:
  `/* Output` F `if recursion depth is even */`;
  foreach $F' \in \textsc{Children}(F)$ **do**
    `enum`(F');
  `/* Output` F `if recursion depth is odd */`;

Proof. Assume that some solutions of $\mathcal{F}_{\min}$ are not outputted by the algorithm, and let F be the smallest one with respect to $\prec$ with $\textsc{Can}(\bar{F}) := f_1, ..., f_\ell$. Let $F_0, ..., F_k = F$ be the canonical path of F. We prove that all F_i, $i < k$ will be outputted by the algorithm. This way, $\textsc{Children}(F_{k-1})$ will be produced, contradicting the fact that F has not been outputted.

Let i be the smallest index such that F_i has not been processed by the algorithm. By minimality of $\bar{F}$ with respect to $\prec$, we know that $\bar{F} \prec \bar{F}_i$. Hence by Lemmas 13 and 14, $f_1, ..., f_i$ is a prefix of $\textsc{Can}(\bar{F}_i)$. So the canonical path of F_i is precisely $F_1, ..., F_i$ and $\textsc{Parent}(F_i) = F_{i-1}$. Now, by minimality of i, we know that F_{i-1} has been outputted by the algorithm, and $F_i \in \textsc{Children}(F_{i-1})$ will be outputted during the processing of F_i.

To obtain a polynomial delaya and polynomial space algorithm, we rely on the fact that the Parent and Childrenfunctions admit a polynomial time complexity. Then with this conditions fulfilled, it remains to show that Algorithm 1 can be implemented so that the delay required to produce a new solution is polynomial in the size of the input and that the memory space is polynomial. Algorithm 1 can be seen as a traversal of the tree of solutions defined by the Parent-Child relation. The time a solution is produced will depend of the height of the solution in the solution tree. It is either produced at the beginning of the call or at the end wheter the height is odd or even. This classic technique in enumeration grant the delay. For this same traversal of the solution tree, Uno [26] developped techniques, that provided the Parent and Children functions admit a polynomial time complexity, will be able to traverse the tree of solutions only using polynomial space. ◀

5 Canonical path reconstruction: a general approach

In fact, the result we proved for chordal completions is part of a more general framework. In this section, we show how to extend the algorithm obtained for chordal graphs, in order to apply it to other graph classes.

As stated in Theorem 3, the authors of [11] proved that if a proximity searchable ordering scheme is prefix-closed, then one can design a polynomial delay and polynomial space algorithm to enumerate $\mathcal{F}_{\max}$ whenever $\mathcal{F}$ is a commutable set system. We prove here that the same remains true even if the system is not commutable. To prove that, we adapt the concept of defined earlier *canonical path reconstruction* in a more general setting to define parent-child relations over $\mathcal{F}_{\max}$ which does not require the commutability condition.

Given $F \in \mathcal{F}$ we denote by F^+ the set $\{x \in \mathcal{U} : F \cup \{x\} \in \mathcal{F}\}$ of candidates for F. Assuming that an ordering scheme π is fixed, we denote by F^i the ith prefix of F according to π.

The notion of prefix-closed ordering has been introduced in [10] and it appears to be a key property to design parent-child relations. Intuitively assume that for each non maximal element of $F \in \mathcal{F}$ we have a preference given by a total ordering $<_F$ over the potential elements that can be added to F (i.e. a total ordering over F^+). This preference depends on the set F and the preference among two elements may vary from one F to another. Then an ordering scheme π is said to be prefix-closed if for each prefix F_i of F the next element f_{i+1} corresponds to the most preferred element that remains in F according to the preference relation $<_{F^i}$. More formally :

▶ **Definition 16.** *An ordering scheme π is* prefix-closed*, if there exists an ordering $<_F$ of the elements of F^+ for each $F \in \mathcal{F}$ that verify the following condition : For every $F \in \mathcal{F}$, $\pi(F) = f_1, ..., f_\ell$ if and only if $f_{i+1} = \min_{<_{F^i}}(F^{i^+} \cap F)$ for any $i < \ell$.*

The goal of this section is to prove the following theorem.

▶ **Theorem 17.** *If π is a polynomial time computable ordering scheme for $\mathcal{F}$ which is both proximity searchable and prefix-closed, then $\mathcal{F}_{max}$ can be enumerated with polynomial delay and polynomial space.*

Notice that any strongly accessible set system has a prefix-closed ordering scheme. Similarly as the one shown for chordal completions one can choose an arbitrary ordering on the elements of $\mathcal{U}$ and simply define $\pi(F) := f_1, ..., f_\ell$ with $f_{i+1} := \min(F^{i^+} \cap F)$. The strong accessibility ensures that this ordering is well defined.

To prove Theorem 17, we define the general parent-child relation over $\mathcal{F}_{\max}$ based on the *canonical path reconstruction method*. Let then π be a prefix-closed, proximity searchable ordering scheme on the set family $\mathcal{F}$.

Whenever an ordering scheme π is proximity searchable, the supergraph of solutions defined by the π-proximity searchable function Neighbours is strongly connected. The goal will be to choose a reference solution F_0 and to identify for each other solution F a canonical path from F_0 to F. The parent of F will be defined as the last solution of this path.

For a prefix-closed ordering scheme π, let us define a total ordering $\prec_\pi$ over $\mathcal{F}_{\max}$. Let $F_1, F_2 \in \mathcal{F}_{\max}$, $\pi(F_1) = t_1, ..., t_{|F_1|}$, $\pi(F_2) = f_1, ..., f_{|F_2|}$ and let $j \geq 0$ be the largest index such that $F_1^j = F_2^j$. Let us denote $F := F_1^j = F_2^j$. Then we have $F_1 \prec_\pi F_2$ if $t_{j+1} <_F f_{j+1}$.

We can now define what will be the canonical path of a solution $F \in \mathcal{F}_{\max}$. To this end, we need to define a function Next : $\mathcal{F}_{\max} \times \mathcal{F}_{\max} \to \mathcal{F}_{\max}$ such that given $F_1, F_2 \in \mathcal{F}_{\max}$ with $F_1 \tilde{\cap} F_2 = i$, $\textsc{Next}(F_1, F_2) \in \{F \in \textsc{Neighbours}(F_1) : F \tilde{\cap} F_2 > i\}$. Since we assumed that π is proximity searchable, we know that the set $\{F \in \textsc{Neighbours}(F_1) \mid F \tilde{\cap} F_2 > i\}$ is not empty. Hence, to define $\textsc{Next}(F_1, F_2)$, one just has to choose deterministically an element of the set $\{F \in \textsc{Neighbours}(F_1) : F \tilde{\cap} F_2 > i\}$. If the function Neighbours produces solutions in a deterministic order, then $\textsc{Next}(F_1, F_2)$ can be chosen as the first $F \in \textsc{Neighbours}(F_1)$ such that $F \tilde{\cap} F_2 > i$. Otherwise and to be as general as possible, let us define $\textsc{Next}(F_1, F_2) := \min_{Lex}\{F \in \textsc{Neighbours}(F_1) : F \tilde{\cap} F_2 > i\}$ as the lexicographically smallest set of $\{F \in \textsc{Neighbours}(F_1) : F \tilde{\cap} F_2 > i\}$.

Given a reference solution F_0, the *canonical path* of $F \in \mathcal{F}_{\max}$ is then defined as the sequence $F_0, ..., F_k$ of elements of $\mathcal{F}$ such that:

- $F_k = F$
- $F_{i+1} = \textsc{Next}(F_i, F)$ for all $i < k$

Each $F \in \mathcal{F}_{\max}$ has a canonical path. Indeed, applying Function Next increases the proximity with F, until F is eventually reached. We define the parent of a solution F of canonical path $F_0, ..., F_k$ as $\textsc{Parent}(F) := F_{k-1}$. The set of children of F is then defined as $\textsc{Children}(F) := \{F' \in \textsc{Neighbours}(F) \mid \textsc{Parent}(F') = F\}$.

▶ **Lemma 18.** *Let $F \in \mathcal{F}_{max}$ and let $F_0, ..., F_k$ be its canonical path. Then for all $i < j \leq k$, $F_j \tilde{\cap} F > F_i \tilde{\cap} F$.*

Proof. By construction of the canonical path of F the proximity increases at each step by at least 1. ◀

The immediate following Corollary ensures that canonical paths are of polynomial size.

▶ **Corollary 19.** *If $F \in \mathcal{F}_{max}$, then its canonical path is of size at most $|F| + 1$.*

▶ **Proposition 20.** *The function PARENT can be computed in time $O(\mathcal{P} + f^2\mathcal{N})$ where f is a maximum size of a solution in $\mathcal{F}_{max}$, $\mathcal{N}$ is the complexity of function NEIGHBOURS and $\mathcal{P}$ is the time needed to compute the function π.*

Proof. To compute the PARENT(F) we first need to compute $\pi(F)$ and then to compute the canonical path of F. By Corollary 19, the path is of length at most f, so we only need to call f times the function NEXT. To compute NEXT(T, F), we compute NEIGHBOURS(T) and for each $F' \in$ NEIGHBOURS(T) we compute the proximity between F' and F and check whether F' is lexicographically smaller than the current best candidate (notice that we don't need to compute $\pi(F')$). Since both operations can be done in $O(f)$, and since |NEIGHBOURS(T)| $< \mathcal{N}$ the function NEXT can be computed in time $O(f\mathcal{N})$. ◀

Since the computation of CHILDREN(F) is done by computing first the set NEIGHBOURS(F) and then filtering it according to the function PARENT, we obtain the following complexity for the function CHILDREN.

▶ **Corollary 21.** *The function CHILDREN can be computed in time $O(\mathcal{P} + f^2\mathcal{N}^2)$ where f is a maximum size of a solution in $\mathcal{F}_{max}$, $\mathcal{N}$ is the complexity of function NEIGHBOURS and $\mathcal{P}$ is the time needed to compute the function π.*

▶ **Lemma 22.** *Let $F \in \mathcal{F}_{max}$ with $\pi(F) = f_1, ..., f_\ell$, let $F_0, ..., F_k$ be its canonical path and let $i \leq k$. If $f_1, ..., f_{F_i \tilde{\cap} F}$ is a prefix of $\pi(F_i)$, then the canonical path of F_i is $F_0, ..., F_i$.*

Proof. Let $T_0, ..., T_h$ be the canonical path of F_i. By definition we have $T_0 = F_0$. Assume that $T_0, ..., T_j = F_0, ..., F_j$ for some $j < i$. By Lemma 18, we know $F_j \tilde{\cap} F < F_i \tilde{\cap} F$, so there exists $r < F_i \tilde{\cap} F$ such that $\{f_1, ..., f_r\} \subseteq F_j$ and $f_{r+1} \notin F_j$.

Hence, since $f_1, ..., f_{r+1}$ is a prefix of $\pi(F_i)$, we have $r = F_j \tilde{\cap} F = F_j \tilde{\cap} F_i < F_i \tilde{\cap} F$. Thus

$$\begin{aligned} F_{j+1} = \text{NEXT}(F_j, F) &= \min_{<_{Lex}} \{F' \in \text{NEIGHBOURS}(F_j) : \{j_1, ..., j_{r+1}\} \subseteq F'\} \\ &= \min_{<_{Lex}} \{F' \in \text{NEIGHBOURS}(T_j) : \{j_1, ..., j_{r+1}\} \subseteq F'\} = \text{NEXT}(T_j, F_i) = T_{j+1}. \end{aligned}$$

This concludes the proof. ◀

The following key lemma is the generalisation of Lemma 13 in the case of chordal completions.

▶ **Lemma 23.** *Let $F_1, F_2 \in \mathcal{F}$ with $\pi(F_1) = f_1, ..., f_\ell$ and $\pi(F_2) = t_1, ..., t_k$. Assume furthermore that there exists $j \leq \ell$ such that $\{f_1, ..., f_j\} \subseteq F_2$, then one of the two possibilities holds:*

1. *$F_2 \prec_\pi F_1$;*
2. *$f_i = t_i$ for all $i \leq j$.*

Proof. Let i be the smallest index such that $t_i \neq f_i$. If $i > j$, then 2 holds and we are done.

Assume that $i \leq j$. By minimality of i, we have $F_1^{i-1} = F_2^{i-1} =: L$. Since $f_i \in F_2$ and since $f_i \in F_1^{i-1^+}$, we know that $f_i \in L^+ \cap F_2$. Therefore, as $t_i = \min_{<_L}(L^+ \cap F_2)$, it holds $t_i \leq_L f_i$, and since $t_i \neq f_i$, we have $t_i <_L f_i$. It proves $F_2 \prec_\pi F_1$. ◀

The previous lemma is the one that makes the canonical path reconstruction method possible. Independently of the prefix-closed property, if one can find a total ordering $\prec$ over $\mathcal{F}_{\max}$ which satisfies Lemma 23, then the canonical path reconstruction method is applicable.

Finally, to prove Theorem 17, it remains to show that the polynomial time computable function Children defines a spanning arborescence on $\mathcal{F}_{\max}$. As already mentioned, the application of the Reverse Search algorithm in [2] would output all solutions without repetition with polynomial delay and polynomial space which will conclude the proof of Theorem 17.

▶ **Theorem 24.** *Children defines a spanning arborescence on $\mathcal{F}_{max}$ rooted at F_0.*

Proof. Recall that the supergraph of solution defined by Function Children is the directed graph on $\mathcal{F}_{\max}$ with arcs set $\{(F_i, F_j) \mid F_j \in \textsc{Children}(F_i)\}$.

Since each $F \in \mathcal{F}_{\max}$, $F \neq F_0$ has only one in-neighbour $\textsc{Parent}(F)$ it is sufficient to show that for all $F \in \mathcal{F}_{\max}$, there exists a path from F_0 to F. Let's denote by $\mathcal{R}(F_0) \subseteq \mathcal{F}_{\max}$ the sets $F \in \mathcal{F}_{\max}$ for which there exists a path from F_0 to F.

Assume for contradiction that $\mathcal{R}(F_0) \neq \mathcal{F}_{\max}$ and let $F := \min_{\prec_\pi}(\mathcal{F}_{\max} \setminus \mathcal{R}(F_0))$. Let $\pi(F) = f_1, ..., f_\ell$ and let $F_0, ..., F_k$ be the canonical path of F. Now, consider the the smallest index $i^* \leq k$ such that $F_{i^*} \notin \mathcal{R}(F_0)$. If $i^* = k$ then $F_{k-1} \in \mathcal{R}(F_0)$ and since $F_{k-1} = \textsc{Parent}(F)$, F would belong to $\textsc{Children}(F_{k-1})$ and then F would belong to $\mathcal{R}(F_0)$. So assume that $i^* < k$.

Let $j := F_{i^*} \tilde{\cap} F$. By minimality of F, we have $F \prec_\pi F_{i^*}$ and since $\{f_1, ..., f_j\} \subseteq F_{i^*}$, by Lemma 23, the $j^{(th)}$ prefix of $\pi(F_{i^*})$ is $f_1, ..., f_j$. Now by Lemma 22 the canonical path of F_{i^*} is $F_0, ..., F_{i^*}$, thus F_{i^*} has F_{i^*-1} as parent. By minimality of i^*, we know that $F_{i^*-1} \in \mathcal{R}(F_0)$ and since $F_{i^*} \in \textsc{Children}(F_{i^*-1})$, we deduce $F_i^* \in \mathcal{R}(F_0)$. ◀

References

1. Stefand Arnbog, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8:277–284, 1987. doi:10.1137/0608024.
2. David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete applied mathematics*, 65(1-3):21–46, 1996.
3. Anne Berry, Jean RS Blair, Pinar Heggernes, and Barry W Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39(4):287–298, 2004.
4. Endre Boros, Benny Kimelfeld, Reinhard Pichler, and Nicole Schweikardt. Enumeration in data management (dagstuhl seminar 19211). Technical report, Dagstuhl Seminar, 2019. doi:10.4230/DagRep.9.5.89.
5. Vincent Bouchitté and Ioan Todinca. Minimal triangulations for graphs with "few" minimal separators. In *European Symposium on Algorithms*, pages 344–355. Springer, 1998.
6. Caroline Brosse, Aurélie Lagoutte, Vincent Limouzy, Arnaud Mary, and Lucas Pastor. Efficient enumeration of maximal split subgraphs and sub-cographs and related classes. *arXiv preprint*, 2020. arXiv:2007.01031.
7. Nofar Carmeli, Batya Kenig, and Benny Kimelfeld. Efficiently enumerating minimal triangulations. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 273–287. ACM, 2017. doi:10.1145/3034786.3056109.

8 Nofar Carmeli, Batya Kenig, Benny Kimelfeld, and Markus Kröll. Efficiently enumerating minimal triangulations. *Discrete Applied Mathematics*, In Press, 2020.

9 Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *Journal of Computer and System Sciences*, 74(7):1147–1159, November 2008. doi:10.1016/j.jcss.2008.04.003.

10 Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. Listing maximal subgraphs satisfying strongly accessible properties. *SIAM Journal on Discrete Mathematics*, 33(2):587–613, 2019.

11 Alessio Conte, Andrea Marino, Roberto Grossi, Takeaki Uno, and Luca Versari. Proximity Search For Maximal Subgraph Enumeration. *arXiv:1912.13446 [cs]*, July 2020. arXiv:1912.13446.

12 Alessio Conte and Takeaki Uno. New polynomial delay bounds for maximal subgraph enumeration by proximity search. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1179–1190, 2019.

13 G. A. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25:71–76, 1961. doi:10.1007/BF02992776.

14 Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 8:171–186, 1976. doi:10.1007/BF01917434.

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

16 Pinar Heggernes and Charis Papadopoulos. Single-edge monotonic sequences of graphs and linear-time algorithms for minimal completions and deletions. *Theoretical Computer Science*, 410(1):1–15, 2009.

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

18 E. Lawler, J. Lenstra, and A. Kan. Generating all maximal independent sets: Np-hardness and polynomial-time algorithms. *SIAM J. Comput.*, 9:558–565, 1980.

19 Tatsuo Ohtsuki. A fast algorithm for finding an optimal ordering for vertex elimination on a graph. *SIAM Journal on Computing*, 5(1):133–145, 1976.

20 Andreas Parra and Petra Scheffler. Characterizations and algorithmic applications of chordal graph embeddings. *Discrete Applied Mathematics*, 79(1-3):171–188, 1997.

21 Noam Ravid, Dori Medini, and Benny Kimelfeld. Ranked enumeration of minimal triangulations. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '19, pages 74–88, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3294052.3319678.

22 Neil Robertson and P.D Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

23 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.

24 Ioan Todinca. *Aspects algorithmiques des triangulations minimales des graphes*. PhD thesis, École normale supérieure (Lyon), 1999.

25 Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A New Algorithm for Generating All the Maximal Independent Sets. *SIAM J. Comput.*, 6(3):505–517, September 1977. doi:10.1137/0206036.

26 Takeaki Uno. Two general methods to reduce delay and change of enumeration algorithms. *National Intsitute of Informatics, Tech. report*, E 4, 2003. URL: http://www.nii.ac.jp/TechReports/public_html/03-004E.pdf.

27 Mihalis Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2:77–79, 1981. doi:10.1137/0602010.