Proximity Search for Maximal Subgraph Enumeration

Alessio Conte, Roberto Grossi, Andrea Marino, Takeaki Uno, Luca Versari

Proximity Search For Maximal Subgraph Enumeration

Alessio Conte
University of Pisa, Italy
conte@di.unipi.it

Roberto Grossi
University of Pisa, Italy
grossi@di.unipi.it

Andrea Marino
University of Florence, Italy
andrea.marino@unifi.it

Takeaki Uno
National Institute of Informatics, Japan
uno@nii.ac.jp

Luca Versari
University of Pisa, Italy and Google Research
veluca@google.com

ABSTRACT

This paper proposes a new general technique for maximal subgraph enumeration which we call *proximity search*, whose aim is to design efficient enumeration algorithms for problems that could not be solved by existing frameworks. To support this claim and illustrate the technique we include output-polynomial algorithms for several problems for which output-polynomial algorithms were not known, including the enumeration of Maximal Bipartite Subgraphs, Maximal k-Degenerate Subgraphs (for bounded k), Maximal Induced Chordal Subgraphs, and Maximal Induced Trees. Using known techniques, such as reverse search, the space of all maximal solutions induces an implicit directed graph called "solution graph" or "supergraph", and solutions are enumerated by traversing it; however, nodes in this graph can have exponential out-degree, thus requiring exponential time to be spent on each solution. The novelty of proximity search is a formalization that allows us to define a better solution graph, and a technique, which we call *canonical reconstruction*, by which we can exploit the properties of given problems to build such graphs. This results in solution graphs whose nodes have significantly smaller (i.e., polynomial) out-degree with respect to existing approaches, but that remain strongly connected, so that all solutions can be enumerated in polynomial delay by a traversal. A drawback of this approach is the space required to keep track of visited solutions, which can be exponential: we further propose a technique to induce a parent-child relationship among solutions and achieve polynomial space when suitable conditions are met. [1]

CCS CONCEPTS

• **Mathematics of computing** → **Graph enumeration**; **Graph algorithms**.

KEYWORDS

graph enumeration, polynomial delay, proximity search

1 INTRODUCTION

Given a universe of elements, such as the vertices or edges of a graph, and a property on them, such as being a clique or a tree,

a listing problem asks to return all subsets of the universe which satisfy the given property.

Listing structures, within graphs or other types of data, is a basic problem in computer science, and it is at the core of data analysis. While many problems can be solved by optimization approaches for the best solution, e.g., by finding the shortest path, or the largest clique, others require finding several solutions to the input problem: in community detection, for example, finding just one "best" community only gives us local information regarding some part of the data, so we may want to find several communities to make sense of the input. Furthermore, many real-world scenarios may not have a clear objective function for the best solution: We may define an algorithm to optimize some desired property, but the optimal solution found may be lacking further properties that emerge during listing or simply not be practical. We may want instead to quickly list several solutions, suitable according to some metrics, then analyze them a posteriori to find the desired one.

In these scenarios, listing only the solutions that are *maximal* under inclusion is a common-sense requirement whenever it can be applied,[2] as maximal solutions subsume the information contained in all others, and can be exponentially fewer: For example, a graph may have up to 2^n cliques, but only $3^{n/3}$ maximal ones [36]. For brevity, we call *maximal listing problem* a listing problem where only the inclusion-maximal solutions should be output.

From a theoretical point of view, listing provides many challenging problems, especially when maximality is required. When dealing with listing algorithms, we are often interested in their complexity with respect to both n, the input size, and N, the size of the output. Algorithms whose complexity can be bounded by a polynomial of these two factors are called *output polynomial* or *polynomial total time* [27]. Interestingly, the hardness of listing problems does not seem to be correlated with that of optimization: there are several NP-hard maximum optimization problems whose corresponding maximal listing problem admits an output-polynomial solution (see, e.g., [1, 46]); on the other hand, there are problems for which one maximal (or maximum) solution can be identified in polynomial time, but an output-polynomial algorithm for listing maximal solutions would imply P=NP [34].

A long-standing question in the area is to find a characterization of which listing problems allow for output-polynomial solutions

---

[1] A preliminary version of this paper has been presented in [16], containing some of the exponential-space algorithms. In this extended version, we factorize the fundamental principles of algorithms in [16] to define a general guide for designing a proximity search algorithm, which we call *canonical reconstruction*. We also introduce a general technique to obtain proximity search algorithms that use polynomial space, as well as proximity search algorithms for new problems, both with exponential-space and polynomial-space requirements. A draft of this extended version is also available at [15].

[2] In other problems, we may want *minimal* solutions instead, although this is usually an equivalent concept, as it corresponds to the complement of a solution being maximal.

and which do not. Furthermore, within output-polynomial algorithms stricter complexity classes exist, such as *incremental polynomial time*, where the time to output the *i*-th solution is polynomial in n and i, and *polynomial delay*, where the time elapsed for outputting the next solution is upper bounded by a polynomial in n. The latter class is of particular interest in practical scenarios, as it guarantees that solutions are output at a regular pace.

In this paper we add a few points to the latter class, by showing that there exist polynomial delay algorithms for some subgraph listing problems. More formally, we prove Theorem 1.1.

Theorem 1.1. *The following problems allow polynomial delay listing algorithms by proximity search:*

PROBLEM	DELAY
maximal induced bipartite sg.	$O(n(m + n\alpha(n)))$
maximal connected induced bipartite sg.	$O(mn)$
maximal bipartite edge-induced sg.	$O(m^3)$
maximal induced k-degenerate sg.	$O(mn^{k+2})$
maximal edge-induced k-degenerate sg.	$O(m^3 n^{k-1})$
maximal induced chordal sg.	$O(m^2 n)$
maximal connected induced chordal sg.	$O(m^2 n)$
maximal edge-induced chordal sg.	$O(m^4 n)$
maximal induced proper interval sg.	$O(m^2 n^3)$
maximal connected induced proper interval sg.	$O(mn^3)$
maximal connected obstacle-free convex hulls	$O(n^4)$
maximal induced trees	$O(m^2)$
maximal connected induced directed acyclic sg.	$O(mn^2)$
maximal connected edge-induced directed acyclic sg.	$O(m^3)$

Where "sg." stands for subgraphs, n and m are the number of vertices and edges, $\alpha(\cdot)$ is the functional inverse of the Ackermann function [44]. All the algorithms use $O(\mathcal{N}n)$ space, where $\mathcal{N}$ is the number of solutions.

To the best of our knowledge, no output-polynomial result was previously known for these problems. For completeness, we consider both *induced* subgraphs (i.e., sets of vertices) and *edge-induced* subgraphs (i.e., sets of edges), as well as the *connected* case where solutions are required to be connected, as the structure of such variants can differ significantly.

Furthermore, we abstract a general technique that can be used to obtain similar results on other problems. We do so by defining a graph whose vertices are the maximal solutions to the listing problem, and with directed edges between pairs of solutions, which we call *solution graph*. The listing problem is solved by traversing the solution graph, and proving that all solutions are found this way. The concept of solution graph is common to existing approaches, and general techniques already exist for building them, e.g., [11]. However, the solution graph built with known approaches such as [11] may have too many edges, resulting in a traversal with exponential delay.

The key concept given in this paper is a technique to build a solution graph with fewer edges, while proving that all solutions are still found by its traversal. An interesting property of this approach is that the resulting algorithms are remarkably simple to implement, while the complexity lies in proving their correctness. We call this technique *proximity search* since at its core lies a problem-specific notion of proximity. This notion acts as a sort of compass on the solution graph built by our algorithm, as given any two solutions S and S^*, we will show that we always traverse an edge from S

to another solution S' that has higher proximity to S^*; as S^* has the highest proximity to itself, this implies that a traversal of the solution graph from any solution finds all others. While others, such as [11, 42], already used the principle of reachability in the solution graph, we aim to define a looser set of necessary condition in order to guarantee this reachability, allowing more freedom in the design of algorithms, while at the same time formalize a technique called *canonical reconstruction* that is effective in decomposing the structure of several problems to fit these rules. The combination of these two parts creates algorithms that overcome the exponential burden imposed by the so-called *input-restricted problem*, a reduced instance of the original problem that dominates the cost per solution of such approaches whose cost may be inherently exponential.

While the space required for a traversal of the graph is inherently proportional to the number of solutions, i.e., can be exponential in n, some output-polynomial techniques such as reverse search are able to work in polynomial space by inducing a tree-like structure on the solution graph, provided that the problem at hand is hereditary (i.e. its property holds for the induced subgraphs) and the input-restricted problem is solvable efficiently.

By adding suitable constraints to the problems considered, we show a technique that combines proximity search with a recent generalization of reverse search to non-hereditary problems [13], obtaining algorithms with both polynomial-delay and polynomial space for some instances of proximity search. In particular, we prove that:

Theorem 1.2. *The following problems allow polynomial delay listing and polynomial space algorithms by proximity search, with the following bounds:*

PROBLEM	DELAY	SPACE
maximal induced bipartite sg.	$O(n^2(m + n\alpha(n)))$	$O(m)$
maximal connected induced bipartite sg.	$O(mn^2)$	$O(m)$
maximal obstacle-free convex hulls	$O(n^4)$	$O(n)$
maximal induced trees	$O(m^2 n^2)$	$O(m)$
maximal induced forests	$O(m^2 n^2)$	$O(m)$

Where notation is as in Theorem 1.1.

1.1 Related Work

The listing problems considered in this paper model solutions as sets of elements (e.g., sets of vertices or edges of a graph), and consist in listing sets of elements with some required property, e.g., inducing a bipartite subgraph, or a tree. We observe that the output is a family of sets, we can associate properties with the corresponding set systems: for example, a property is hereditary when each subset of a solution is a solution, which corresponds to the well-known independence systems [34].

In this context, a simple yet powerful technique is recursively partitioning the search space into all solutions containing a certain element, and all that do not. This technique, usually called binary partition or simply backtracking, proves efficient when listing all solutions [40], and can be used to design algorithms that are fast in

practice,[3] or that can bound the number of solutions in the worst-case [21]. On the other hand, this strategy rarely gives output-polynomial algorithms when dealing with maximal solutions, as we may spend time exploring a solution subspace that contains many solutions but no maximal one.

To obtain output-polynomial algorithms for maximal solutions, many algorithms rely on the following idea: given a maximal solution S, and some element $x \notin S$, the hardness of listing solutions *maximal within $S \cup \{x\}$* is linked to the hardness of listing them in a general instance. One of the earliest mentions of the idea can be found in the seminal paper by Lawler et al. [34], that generalizes ideas from Paull et al. [38] and Tsukiyama et al. [46], and has been formally defined as *input-restricted problem* by Cohen et al. [11].

The intuition is that the solutions obtained this way, using a maximal solution S and an element not in S, can be used to generate new maximal solutions of the original problem. We can thus traverse an implicit directed graph, which we will call *solution graph*, where the vertices are the maximal solutions and the out-neighbors are obtained by means of the input-restricted problem.

In particular, [34] showed how solving this problem could yield an output-polynomial and polynomial space listing algorithm for properties corresponding to *independence systems*, assuming the input-restricted problem has a bounded number of solutions. [11] showed that the strategy could be extended to the more challenging *connected-hereditary* graph properties (i.e., where *connected* subsets of solutions are solutions) using exponential space, and recently, [13] showed that the same result can be obtained in polynomial space for *commutable* set systems (which include connected-hereditary properties).

A clear limitation of this approach is that, in order to obtain polynomial-delay algorithms, the input-restricted problem needs to be solved in polynomial time. This is possible for some problems (e.g., cliques and independent sets), but impossible for others, simply because their input-restricted problems may have exponentially many solutions. Figure 1 shows an example for maximal bipartite subgraphs.

The literature contains many more results concerning the enumeration of maximal/minimal solutions, e.g., [1, 8, 23, 24, 31, 42], and in particular regarding challenging problems such as the well-known minimal hypergraph transversals/dominating sets problem [19, 25, 28]. However, to the best of our knowledge, the only two effective general techniques for listing maximal solutions in an output-sensitive fashion are the extension problem (binary partition, flashlight search), and the input-restricted problem: proximity search can be a valuable tool when the previous two fail. We motivate this by showing the first polynomial delay algorithms for several maximal listing problems whose associated input-restricted problem is not solvable in polynomial time.

As mentioned above, a preliminary version of this paper containing some of the exponential-space algorithms has appeared in [16]. Since its publication, some preprints [6, 7, 33] have appeared that apply the technique to obtain new output-polynomial algorithms. In particular, [7] solves the enumeration of Maximal Induced Interval Subgraphs by proposing some variations to proximity search [16].

---

[3]E.g., implementations of the Bron-Kerbosh [45] algorithm tend to be faster than those of output-polynomial algorithms [12] for listing maximal cliques.

1.2 Overview

The main contribution of the paper is presenting *proximity search*, a general technique that can be used to solve several enumeration problems in polynomial delay, and *canonical reconstruction*, a way to design a proximity search algorithm by exploiting orderings of solutions of the problem at hand. exponentially many solutions.

By using this technique we show polynomial delay algorithms for several maximal listing problems such as maximal bipartite subgraphs and the others mentioned in Theorem 1.1. Other than providing efficient algorithms, we remark that the technique may help gain further insight on which classes of problems allow output-polynomial listing algorithms and which do not.

The paper is organized as follows: First, we introduce some basic concepts and notation in Section 2. We then explain the proximity search technique, and formally define a class of problems, called *proximity searchable*, which allow for a polynomial delay algorithm by its application.

Generality comes sometimes at the expense of efficiency but allows for a more intuitive understanding of the concepts at hand. For this reason, we divide the explanation in two parts: the first one, in Section 3, formalizes the constraints required for a proximity search algorithm. The second, Section 4, introduces a technique which we call *canonical reconstruction* for implementing proximity search. While canonical reconstruction is not the only way to obtain a proximity search algorithm, we observed that is often a powerful and elegant way to model the problem at hand.

Following, Sections 5-10, shows how to prove that the problems in Theorem 1.1 are proximity searchable and thus allow polynomial-delay algorithms.

As a drawback of the above algorithms is an exponential space requirement, we then propose a technique to address this issue, when suitable conditions are met: define a parent-child relation between solutions, in the style of reverse-search, as detailed in Section 11, and give the algorithms in Section 12. The resulting bounds are shown in Theorem 1.2. While this technique does not apply to all problems in Theorem 1.1, when it does it allows us to obtain polynomial-delay and polynomial-space algorithms for several problems whose input-restricted problem cannot be solved in polynomial time, including non-hereditary ones.

2 PRELIMINARIES

Most of the enumeration problems addressed in this paper consider a simple undirected graph G, whose vertex set is denoted as $V(G)$ and edge set as $E(G)$, or simply $G = (V, E)$ when it is clear from the context. The neighborhood of a vertex v is denoted as $N(v)$. For brevity, we refer to $|V(G)|$ as the number n of vertices, to $|E(G)|$ as the number m of edges, and to the maximum degree of a vertex in G as $\Delta = \max_{v \in V} |N(v)|$. Furthermore, we assume the vertices to be labeled arbitrarily in increasing order $v_1, \ldots, v_n$, and say that v_i is smaller than v_j if $i < j$. We say that a neighbor of v_i is a *forward* neighbor if it comes later than v_i in the order, and a *backward* neighbor otherwise.

For a set of vertices $A \subseteq V(G)$, $E[A]$ denotes the edges of G whose endpoints are both in A, and $G[A]$ the graph $(A, E[A])$, i.e., the subgraph induced in G by A. Similarly, for a set B of edges, $V[B]$ denotes the vertices incident to an edge in B and $G[B] = (V[B], B)$.
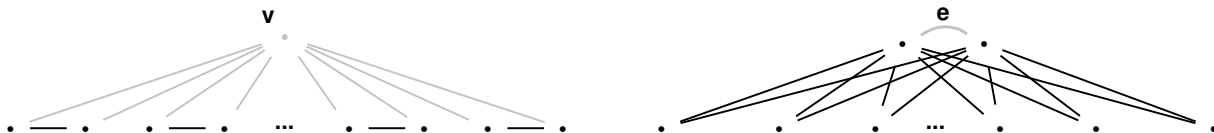
Figure 1: Instances of input-restricted problem for maximal bipartite subgraphs. On the left: the black dots define a maximal bipartite induced subgraph; adding the vertex v creates a graph with exponentially many maximal induced bipartite subgraphs, as we can obtain one by removing a vertex from each connected pair in the bottom in any combination. On the right: the black edges define a maximal bipartite subgraph, and the addition of edge e creates a graph with exponentially many maximal edge-induced bipartite subgraphs: every vertex on the bottom is incident to two edges; removing exactly one for each vertex yields a maximal edge-induced bipartite subgraph.

As common in the literature, we call *induced subgraphs* those of the former kind, defined by a set of vertices, and *edge-induced subgraphs* (or simply subgraphs) those of the latter, defined by a set of edges. When dealing with subgraphs defined by a set of vertices (resp. edges) A, we will sometimes use A to refer to both the vertex set (resp. edge set) and the subgraph $G[A]$ it induces, when this causes no ambiguity. We will also use $\mathrm{cc}_v(A)$ to refer to the *connected component* of $G[A]$ which includes the vertex v. For further notation, we refer to the standard terminology in [18].

For a set of vertices $A \subseteq V(G)$ which corresponds to a solution of the problem at hand, we say that A is *maximal* if there is no $A' \subseteq V(G)$ such that $A' \supset A$ and A' is also a solution. While not strictly necessary for the proximity search technique, in the following we will often rely on a simple "maximalization" function, named $\mathrm{comp}(A)$: this function takes a (not necessarily maximal) solution A and "completes" it, returning some maximal solution $A' \supseteq A$. We will refer to the computational cost of this function as C_t. Note that it is always possible to devise a polynomial-time computable $\mathrm{comp}(\cdot)$ function for hereditary and connected-hereditary properties where solutions can be recognized in polynomial time, by simply trying to add vertices until no longer possible [11].

For simplicity, we disregard the presence of isolated vertices in the complexity analysis of the algorithms provided: these are trivially handled for the problems considered in this paper (either they can all be added "in bulk" to every solution, as for bipartite subgraphs, or each constitutes a maximal solution by itself, as for connected bipartite subgraphs), can be removed with an $O(n)$ time preprocessing; this means we are able to perform operations like a visit of the graph in $O(m)$ time rather than $O(m + n)$ time.

3 PROXIMITY SEARCH OUTLINE

Proximity search is based on traversing an implicit solution graph, where the vertices are all the solutions to be listed and each directed arc goes from a solution to another using a neighboring function. Several solution graphs are possible, depending on how the neighboring function is defined. Apart from the fact that the resulting solution graphs are not necessarily strongly connected and some care should be taken to list all the solutions, the main hurdle is that the degree of the solution graphs can be exponential (as the number of solutions can be exponentially large in the input size), thus preventing to achieve polynomial delay when running a simple traversal. Proximity search circumvents these issues by designing a suitable neighboring function, denoted NEIGHBORS$(\cdot)$,

that guarantees that the resulting solution graph it implicitly defines is strongly connected and of polynomial degree. Both these properties cannot be guaranteed with the current state of the art for a number of problems discussed later.

We devote this section to formalize the general structure of proximity search, and the class of problems to which the technique can be applied. Also, we introduce the notion of proximity, symbolized by $\tilde{\cap}$, to act as a sort of oracle for navigating the solution graph.

For reference in what we discuss next, we give the pseudo-code of the generic traversal of a solution graph based on the NEIGHBORS$(\cdot)$ function, as shown in Algorithm 1. As noted earlier, the algorithms obtained by specializing this generic traversal are remarkably simple: In a depth-first search traversal where the set S keeps track of just the last visited solution, we only need to implement the NEIGHBORS$(\cdot)$ function. On the other hand, the complexity is mostly hidden behind proving their completeness: Notably, the very notion of proximity $\tilde{\cap}$ is only used in the proofs, and never actually appears in Algorithm 1.

Algorithm 1: Traversal of the solution graph by proximity search.

> **input** : Graph $G = (V, E)$ and listing problem $\mathcal{P}$
> **output**: All (maximal) solutions of $\mathcal{P}$ in G
> **global** : Set $\mathcal{S}$ of solutions found, initially empty

1 $S \leftarrow$ an arbitrary solution of $\mathcal{P}$ /* (e.g. comp($\emptyset$)) */
2 Call ENUM(S)

3 **Function** *ENUM*(S)
4   Add S to $\mathcal{S}$
    /* Output S if recursion depth is even */
5   **foreach** $S' \in$ *NEIGHBORS*(S) **do**
6          $\lfloor$ **if** $S' \notin \mathcal{S}$ **then** ENUM(S')
    /* Output S if recursion depth is odd */

In order to start the algorithm, we need one arbitrary maximal solution S. We remark that identifying one maximal (not maximum) solution is typically trivial, and can be achieved for example by running $\mathrm{comp}(\emptyset)$ when the $\mathrm{comp}(\cdot)$ function is computable in polynomial time.

We formally define the class of problems which allow for a polynomial delay algorithm using this structure as *proximity searchable*.

Definition 3.1 (Proximity searchable). Let $\mathcal{P}$ be a listing problem over a universe $\mathcal{U}$ with set of solutions $\mathcal{S} \subseteq 2^{\mathcal{U}}$, where each solution is a subset of the universe. $\mathcal{P}$ is *proximity searchable* if there exists a proximity function $\tilde{\cap} : \mathcal{S} \times \mathcal{S} \rightarrow 2^{\mathcal{U}}$ and a neighboring function NEIGHBORS$(\cdot) : \mathcal{S} \rightarrow 2^{\mathcal{S}}$, such that the following holds:

(1) One solution of $\mathcal{P}$ can be identified in time polynomial in $|\mathcal{U}|$.
(2) NEIGHBORS$(\cdot)$ is computable in time polynomial in $|\mathcal{U}|$.
(3) Given any two distinct solutions $S, S^* \in \mathcal{S}$, there exists $S' \in$ NEIGHBORS(S) such that $|S' \tilde{\cap} S^*| > |S \tilde{\cap} S^*|$.

The above conditions imply the following one, which is reported for the sake of clarity.

(4) For any fixed S^*, $|S \tilde{\cap} S^*|$ is maximized for (and only for) $S = S^*$.

If a problem is proximity searchable, then it is straightforward to see that we obtain a polynomial delay algorithm for it by using the corresponding NEIGHBORS$(\cdot)$ function in Algorithm 1. Let us formally prove it.

THEOREM 3.2. *All proximity searchable listing problems have a polynomial delay listing algorithm.*

PROOF. We first show that if a NEIGHBORS$(\cdot)$ function satisfies Definition 3.1, the implicit solution graph it induces is strongly connected. Given any two distinct solutions $S, S^* \in \mathcal{S}$, we know by Definition 3.1.(3) that there exists $S' \in$ NEIGHBORS(S) such that $|S' \tilde{\cap} S^*| > |S \tilde{\cap} S^*|$. By induction on S and S', it follows that we will eventually reach a solution S that globally maximizes $|\cdot \tilde{\cap} S^*|$, which by Definition 3.1.(4) is precisely S^*.

Based on the above properties, we next show that Algorithm 1 outputs (all and only) the solutions of any proximity searchable problem with no duplication. Firstly, Algorithm 1 returns only maximal solutions, as it only outputs the initial maximal solution found on line 1, found polynomial time by Definition 3.1.(1), and the output of calls to NEIGHBORS(S) which contain maximal solutions.

We say that a solution is *visited* when ENUM(S) is called. In Algorithm 1 all solutions added to $\mathcal{S}$ are visited at most once, thanks to the membership test in the set $\mathcal{S}$; this guarantees that the same solution is never output twice. As the graph defined by NEIGHBORS(S) is strongly connected, the traversal done by Algorithm 1 starting from the solution found on line 1 must find all solutions.

To complete the proof, we show that Algorithm 1 runs in polynomial delay.

Firstly, NEIGHBORS(S) requires polynomial time by Definition 3.1.(2) and thus can only return a polynomial number of solutions; this means the out-degree of every node in the implicit solution graph is polynomial and we can iterate over it in polynomial time.

As a new recursive call is performed only when a new solution is found, the amortized cost per solution is bound by the cost of a recursive call, i.e., the cost of lines 4–6. As the cost of NEIGHBORS$(\cdot)$ is polynomial, and $\mathcal{S}$ can be easily maintained in polynomial time (in Appendix A we show this latter cost to be negligible for all algorithms presented here), it follows that the amortized cost per solution is polynomial. In order to get polynomial delay, we can employ the *alternative output* [47] method, that can be applied to any recursive algorithm that outputs a solution in each recursive

call: by performing output in pre-order when the recursion depth is even, and post-order when it is odd, the delay will be bounded by that of a constant number of recursive calls, i.e., polynomial. □

The following observations are in order:

- $\tilde{\cap}$ is not actually used in Algorithm 1, and does *not* need to be computed.
- Proximity search can be applied to all listing domains where solutions are modeled by set systems, not just graphs.
- Proximity search is mainly intended for maximal listing problems, however, it is not strictly limited to it.
- Maximal listing problems in which the input-restricted problem is computable in polynomial time (as well as the COMP$(\cdot)$ function) are proximity searchable.[4]
- The polynomiality constraint on NEIGHBORS$(\cdot)$ can be relaxed: it can be trivially seen how computing NEIGHBORS$(\cdot)$ in Incremental Polynomial Time (resp. Polynomial Total Time) yields and Incremental Polynomial Time (resp. Polynomial Total Time) algorithm.
- The cost per solution and delay of the algorithm is the complexity of the NEIGHBORS$(\cdot)$ function (we show in Appendix A how maintaining $\mathcal{S}$ is negligible).

In the rest of the paper, we show how to suitably model several problems to obtain new polynomial-delay algorithms for problems that, to the best of our knowledge, could not be previously solved in polynomial delay. We show these algorithms by providing suitable $\tilde{\cap}$ and NEIGHBORS$(\cdot)$ functions, proving that they satisfy Definition 3.1, which automatically give us a polynomial delay listing algorithm by Algorithm 1.

We will use a common notation: S is an arbitrary solution, and S^* the "target" solution.

4 PROXIMITY SEARCH BY CANONICAL RECONSTRUCTION

We make concrete use of the abstract notion of proximity search and introduce a technique, which we call *canonical reconstruction*. While it is kept separate from the previous section for cohesiveness, we find this technique to often be the right way to look at maximal subgraph listing problems. Since we deal with graphs, the universe $\mathcal{U}$ is the vertex set, unless explicitly specified.

To accompany the explanation, we detail its implementation in the case of Maximal Connected Induced Bipartite Subgraphs in Section 5.

The technique is based on the definitions of *canonical order* and *canonical extender* for solutions, which depend entirely on the problem at hand, and it is intuitively a way to harness its structure.

EXAMPLE. *For a Maximal Connected Induced Bipartite Subgraph S, we will use as canonical order a BFS-order of $G[S]$ starting from its vertex of smallest id, where ties are broken by vertex id: For the subgraph in Figure 2 (b) this order is 2, 3, 5, 8, 11, 7, 10, and for the one in (c) it is 2, 3, 8, 12, 11.*

---

[4]In essence, we obtain as a special case the same solution graph as known algorithms based on the input-restricted problem [11, 34, 46]: For a solution S we compute NEIGHBORS(S) by sequentially taking all elements $v \in \mathcal{U} \setminus S$, solving the input-restricted problem for $S \cup \{v\}$, and applying COMP$(\cdot)$ on the results.
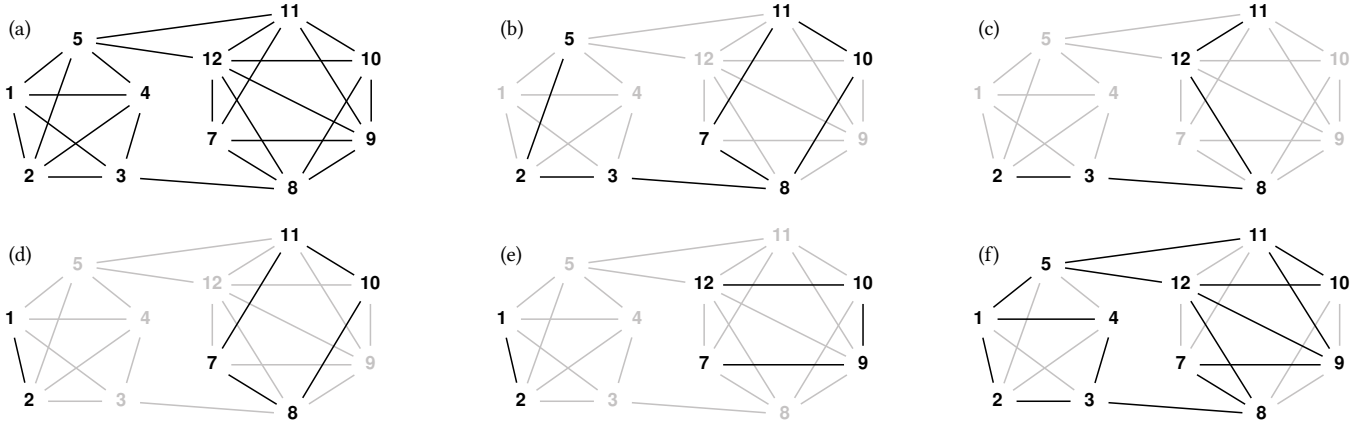
Figure 2: a: a graph. b,c: two maximal connected induced bipartite subgraphs of (a). d,e: two maximal induced bipartite subgraphs of (a). f: a maximal edge-induced bipartite subgraph of (a).

Canonical order and proximity. Simply assume that each solution S is given an ordering $s_1, \ldots, s_{|S|}$ of its elements which will satisfy some problem-specific conditions. We require that any *prefix* $s_1, \ldots, s_i$ of this order corresponds to a (non-maximal) solution $\{s_1, \ldots, s_i\}$. In the rest of the paper, we will refer as prefix of the order to both the sequence $s_1, \ldots, s_i$ and the corresponding set of elements $\{s_1, \ldots, s_i\}$. Note that the ordering is *not* required to be efficiently computable, as the proximity search algorithm never actually computes it: it is only used in the correctness proof of the neighboring function. Moreover, the ordering is adaptive to each solution, so the same elements can be ranked differently in distinct solutions.

Given the order, we define the proximity function $\tilde{\cap}$ as follows.

Definition 4.1 (proximity). Given two solutions S and S^*, let $s_1^*, \ldots s_{|S^*|}^*$ be the canonical order of S^*: the proximity $S \tilde{\cap} S^*$ between S and S^* is the longest *prefix* $s_1^*, \ldots, s_i^*$ of the canonical order of S^* whose elements are all contained in S.

It should also be noted that the operation is *not* symmetric, i.e., we may have $S \tilde{\cap} S^* \neq S^* \tilde{\cap} S$.

EXAMPLE. *Let S be the subgraph shown in Figure 2 (b) and S^* the one shown in (c). Considering the canonical orders mentioned above, we can see that $S \tilde{\cap} S^* = \{2, 3, 8\}$, while $S^* \tilde{\cap} S = \{2, 3\}$.*

Canonical extender. The goal of a proximity search algorithm is to exploit Definition 3.1.(3): given S, for any S^*, find some S' such that $|S' \tilde{\cap} S^*| > |S \tilde{\cap} S^*|$. Using Definition 4.1, $S \tilde{\cap} S^*$ is a prefix $s_1^*, \ldots, s_i^*$ of the canonical order of S^*, so we want to find any solution S' that contains a longer prefix, i.e., $s_1^*, \ldots, s_{i+1}^*$ (possibly ordered differently and interspersed in the canonical order of S'). Since we must at least add the vertex s_{i+1}^*, we call s_{i+1}^* the *canonical extender* of S, S^*. Armed with this notion, we want to proceed conceptually as follows for a given solution S.

(1) Guess which node $v \notin S$ is the canonical extender s_{i+1}^* (try all possibilities, n at most).
(2) Guess a *removable set* $X \subseteq S$ from $S \cup \{v\}$, i.e., such that $S \setminus X \cup \{v\}$ is a solution and $X \cap \{s_1^*, \ldots, s_i^*\} = \emptyset$.
(3) Obtain S' as the outcome of $\mathrm{COMP}(S \setminus X \cup \{v\})$.

In essence, we want to add s_{i+1}^* to S, then turn the result back into a solution by removing some elements, but without affecting the proximity $s_1^*, \ldots, s_i^*$.

Recalling that prefixes of a canonical order are required to be (non-maximal) solutions, indeed $s_1^*, \ldots, s_{i+1}^*$ is a solution; hence, a removable set X always exists (e.g., $X = S \setminus \{s_1^*, \ldots, s_i^*\}$). The key point is that we want to satisfy the proximity requirement for all S^* (that can be exponentially many) using only a *polynomial* number of removable sets X. While there is no general rule for this, and indeed, solving this for some problems would imply P=NP, we will observe in this paper how it is possible to do so in some cases where a canonical order can efficiently decompose the underlying structure of the solution.

Canonical reconstruction. Now we have all the ingredients to formalize below the required structure for adopting our strategy.

Definition 4.2. (Proximity search by canonical reconstruction) Given a maximal listing problem $\mathcal{P}$, in which each maximal solution S is associated with a canonical ordering $s_1, \ldots, s_{|S|}$, we say that $\mathcal{P}$ admits a *canonical reconstruction* if the following holds.

(1) Any prefix $s_1, \ldots, s_i$ of the canonical order of any maximal solution S is a (non-maximal) solution of $\mathcal{P}$.
(2) Given a maximal solution S and any $v \notin S$, there is set $\mathcal{X} \subseteq 2^S$ of *removables*, such that
  - $\mathcal{X} = \{X_1, X_2, \ldots\}$ can be computed in polynomial time.
  - $S \setminus X_i \cup \{v\}$ is a solution of $\mathcal{P}$ for any $X_i \in \mathcal{X}$.
  - For any S^* such that v is the canonical extender of S, S^*, there is at least one $X_i \in \mathcal{X}$ such that $(S \tilde{\cap} S^*) \cap X_i = \emptyset$. [5]
(3) There is a polynomial-time computable function $\mathrm{COMP}(A)$ which takes a solution A of $\mathcal{P}$ and returns a maximal solution $A' \supseteq A$ of $\mathcal{P}$.

We further define the *canonical reconstruction function*, as

$$\mathrm{NEIGHBORS}(S, v) = \bigcup_{X_i \in \mathcal{X}} \mathrm{COMP}((S \setminus X_i) \cup \{v\})$$

---

[5]Indeed, $(S \tilde{\cap} S^*) \cap X_i = \emptyset$, i.e., X_i does not intersect the proximity, implies that $S \setminus X_i \cup \{v\}$ contains $(S \tilde{\cap} S^*) \cup \{v\}$, which extends the proximity with v.

and this corresponds to the solutions $S' \in$ NEIGHBORS(S) for which v is the canonical extender of S, S'. Hence, NEIGHBORS(S) is obtained as $\bigcup_{v \in V(G)}$ NEIGHBORS(S, v).[6]

We observe how the removables and the neighboring function can be derived from one another, so the algorithm can be defined by providing either one: one can focus on defining removables from $S \cup \{v\}$ that do not intersect the proximity, or equivalently solutions contained in $S \cup \{v\}$ that fully contain the proximity.

We also recall that a polynomial-time computable function COMP$(\cdot)$ trivially exists for all hereditary and connected-hereditary properties that can be recognized in polynomial time (these include all the example problems shown in this paper).

Finally, we show how canonical reconstruction immediately implies the maximal listing problem at hand is proximity searchable, using the NEIGHBORS$(\cdot)$ function defined above:

THEOREM 4.3. *All maximal listing problems that allow a canonical reconstruction are proximity searchable.*

PROOF. Let us show that a listing problem $\mathcal{P}$ that satisfies Definition 4.2 satisfies the four conditions of Definition 3.1. Condition (1) is trivially satisfied, say, using COMP$(\emptyset)$. As for condition (2), recall that

$$\text{NEIGHBORS}(S) = \bigcup_{v \in V(G)} \bigcup_{X_i \in \mathcal{X}} (\text{COMP}(S \setminus X_i \cup \{v\})).$$

Considering that both $|V(G)|$ and $|\mathcal{X}|$ are polynomial, and COMP$(\cdot)$ takes polynomial time, it follows that computing NEIGHBORS(S) takes polynomial time.

For condition (3), consider the canonical extender v for S, S^*: By Definition 4.2 there is $X_i \in \mathcal{X}$ such that $X_i \cap (S \tilde\cap S^*) = \emptyset$; it holds that $S' = \text{COMP}(S \setminus X_i \cup \{v\}) \in \text{NEIGHBORS}(S, v) \subseteq \text{NEIGHBORS}(S)$, and $(S \tilde\cap S^*) \cup \{v\} \subseteq S'$, thus $|S' \tilde\cap S^*| > |S \tilde\cap S^*|$ because of v.

Finally, condition (4) is satisfied by looking at the definition of proximity in Definition 4.1: fixed S^*, the proximity $S \tilde\cap S^*$ is maximized if $S \supseteq S^*$, as we have $S \tilde\cap S^* = S^* = S$; however, as this is a maximal listing problem, all solutions are inclusion-wise maximal, meaning that $S \supseteq S^*$ is only true for $S = S^*$. □

As a final remark, we note a somewhat surprising feature of this technique: while in general connected-hereditary properties (e.g., Maximal Connected Induced Bipartite Subgraphs) are more challenging to deal with than hereditary ones (e.g., Maximal Induced Bipartite Subgraphs), in the case of proximity search there is typically no difference, and in some instances, we even use the connected case as a starting point for the non-connected one (see, e.g., Section 5.2).

5 MAXIMAL BIPARTITE SUBGRAPHS

We now illustrate how to apply proximity search to maximal bipartite subgraph enumeration, giving the full details for the example in the previous section.

A graph G is bipartite if its vertices can be partitioned into two sets V_0, V_1, such that $V_0 \cap V_1 = \emptyset$, $V_0 \cup V_1 = V(G)$, and both $G[V_0]$ and $G[V_1]$ are edge-less graphs. Equivalently, G is bipartite if it has no cycle of odd length. Maximal bipartite subgraphs have also

been studied as *minimal odd cycle transversals* [32], as one is the complement of the other.

The problem of listing *all* bipartite (and induced bipartite) subgraphs has been efficiently solved in [48]. However, to the best of our knowledge, neither the techniques in [48] nor other known ones extend to efficiently listing *maximal* bipartite subgraphs, which poses a challenge. Consider the instance of input-restricted problem shown in Figure 1 (left). We can exploit the fact that a subgraph of a bipartite graph is itself bipartite, meaning that the property is hereditary. Hence, we could take the current solution S (which are the endpoints of the bold edges) and a vertex $v \notin S$, to then try to list all the maximal solutions contained in the induced subgraph $G[S \cup \{v\}]$; however, $G[S \cup \{v\}]$ has exponentially many solutions, meaning we cannot solve the input-restricted problem in polynomial time and thus we cannot get polynomial delay with the techniques from [11, 13, 34]. The best we could hope for is solving the input-restricted problem in polynomial delay or incremental polynomial time, which would yield an incremental polynomial time algorithm for the general problem [11]. Figure 1 (right) shows an analogous situation for edge-induced subgraphs.

We thus turn to proximity search. First, let us introduce some preliminary notions: We denote an induced bipartite *subgraph* of G as a pair of vertex sets $\langle B_0, B_1 \rangle$, with $B_0 \cap B_1 = \emptyset$ and $B_0 \cup B_1 \subseteq V(G)$, such that $G[B_0]$ and $G[B_1]$ are edge-less graphs. By convention, B_0 is the side of the bipartition containing the vertex of smallest label among those in the subgraph. In case $G[B_0 \cup B_1]$ has multiple connected components, this applies to all components. This way, any bipartite subgraph (connected or not) always has a *unique* representation $\langle B_0, B_1 \rangle$. We will sometimes use simply B to refer to the subgraph $G[B_0 \cup B_1]$ induced by $\langle B_0, B_1 \rangle$. When performing COMP(B) (defined at the end of Section 2) and B is not connected, this may move some vertices from B_0 to B_1 and vice versa due to different components becoming connected; even when B is connected, if a vertex with smaller label than all others in B is added to B_1, then B_0 and B_1 are immediately swapped to preserve the invariant of the smallest vertex being in B_0. We define the intersection between two bipartite subgraphs B and B' as the set of all shared vertices, i.e.: $B \cap B' = (B_0 \cup B_1) \cap (B'_0 \cup B'_1)$.

We consider the case of connected induced bipartite subgraphs in Section 5.1. We will later briefly show how this structure can be adapted to cover the non-connected and non-induced cases with small changes in Sections 5.2 and 5.3. Their complexity will be discussed in Section 5.4.

5.1 Listing Maximal Connected Induced Bipartite Subgraphs

Let $B = \langle B_0, B_1 \rangle$ be a maximal induced bipartite subgraph of G, and v a vertex not in B, i.e., in $v \in V(G) \setminus B$. Looking at Definition 4.2, we immediately observe that a polynomial-time computable function COMP$(\cdot)$ exists since the problem is connected-hereditary. Then, we need to define a suitable canonical order, and prove the existence of the corresponding removables.

Consider a BFS order of $G[B]$ starting from its vertex of smallest label, say b_1. In this order, a vertex u precedes a vertex v if the distance of u from b_1 is smaller than that of v or, in case the distance is equal, u's label is smaller than v's.

[6]For completeness, we define NEIGHBORS$(S, v) = \{S\}$ for $v \in S$, as $S \cup \{v\} = S$ is already a solution of $\mathcal{P}$.

$S = \{2, 3, 5, 7, 8, 10, 11\}$, with $S_0 = \{2, 8, 11\}$, $S_1 = \{3, 5, 7, 10\}$, and $v = 12$

$X_0 = N(v) \cap S_0 = \{8, 11\}$, $X_1 = N(v) \cap S_1 = \{5, 7, 10\}$

$S \setminus X_1 \cup \{v\} = \{2, 3, 8, 11, 12\}$ (top)

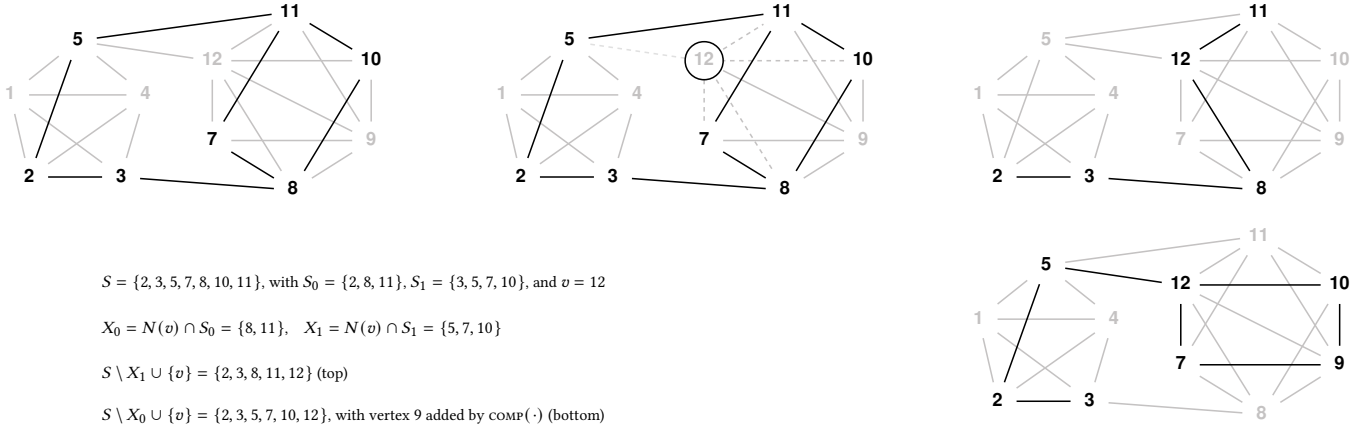$S \setminus X_0 \cup \{v\} = \{2, 3, 5, 7, 10, 12\}$, with vertex 9 added by comp$(\cdot)$ (bottom)

Figure 3: The steps taken by the neighboring function neighbors(S, v)**, for a possible solution S of the graph in Figure 2 and $v = 12$. The bottom two lines correspond to the neighboring solutions produced by the function** neighbors$(S, 12)$**.**

Definition 5.1 (canonical order for connected induced bipartite subgraphs). The *canonical* order of a *connected induced bipartite subgraph B* is the sequence $b_1, \ldots, b_{|B|}$ given by a BFS order of $G[B]$ rooted at the vertex b_1 of smallest label, where ties are broken by placing the vertex of smallest label first.

For the subgraph in Figure 2 (b) the canonical order is 2, 3, 5, 8, 11, 7, 10, and for the one in (c) it is 2, 3, 8, 12, 11. The definition of proximity is then automatically given by Definition 4.1.

Last ingredient for Definition 4.2 is the set $\mathcal{X} \subseteq 2^B$, that contains just *two* removables. In order to get a bipartite graph, it is possible to make two removables as follows:

- $X_0 = N(v) \cap B_0$,
- $X_1 = N(v) \cap B_1$.

That is, remove all the neighbors of v in one of the two sides B_i: clearly, v can be included in B_i as it is now only adjacent to vertices of B_{1-i}. While this works for the Maximal Induced Bipartite Subgraphs problem, we have the further constraint of *connectivity*, so we must also discard every vertex that is not in the same connected component as v. The removables become as follows:

- $X_0 = B \setminus cc_v(\{v\} \cup (B_0 \setminus N(v)) \cup B_1)$,
- $X_1 = B \setminus cc_v(\{v\} \cup (B_1 \setminus N(v)) \cup B_0)$.

That is, we remove all vertices not in the same connected component as v, after introducing v and removing all its neighbors in either B_0 or B_1.

We can use these to create the neighboring function to be plugged in Algorithm 1, following Definition 4.2:

Definition 5.2 (neighboring function for maximal connected induced bipartite subgraphs).

neighbors$(B, v) = \{$comp$(cc_v(\{v\} \cup (B_i \setminus N(v)) \cup B_{1-i})) \mid i = 0, 1\}$

A graphical example of this procedure is given in Figure 3 (for simplicity, we adopt an example where the subgraphs are connected after removing v's neighbors, so the removables are equivalent to the simpler ones of the non-connected version).

Lemma 5.3. *The problem of listing all Maximal Connected Induced Bipartite Subgraphs admits a canonical reconstruction.*

Proof. For the canonical order given in Definition 5.1, any prefix induces a graph that is connected because of the BFS order, and bipartite because bipartite subgraphs are hereditary, so condition (1) of Definition 4.2 is satisfied. As for condition (2), it is evident from the definition of removables (alternatively, of the neighboring function) that they can be computed in polynomial time, and that they produce connected bipartite subgraphs. We only need to show that the third item holds: given B, B^* and their canonical extender $\dot{v}$, we have $B \tilde{\cap} B^* \cap X_i = \emptyset$ for either $i = 0$ or $i = 1$; this will imply that $|B' \tilde{\cap} B^*| > |B \tilde{\cap} B^*|$ for $B' = $comp$(cc_{\dot{v}}(\{\dot{v}\} \cup (B_i \setminus N(\dot{v})) \cup B_{1-i}))$, so the proximity is successfully increased.

If $B \tilde{\cap} B^* = \emptyset$ the claim is trivially true, as we can consider b_1^* as canonical extender. Otherwise, let $Z = B \tilde{\cap} B^* = \{b_1^*, \ldots, b_h^*\}$, and we have $\dot{v} = b_{h+1}^*$. By Definition 5.1, Z is a connected induced bipartite subgraph, meaning that it allows a unique bipartition Z_0, Z_1 (with Z_0 being the set containing the vertex of smallest label in Z, that is, b_1^*). Since b_1^* is the vertex of smallest label in B^*, it will be in B_0^*, so it follows that $Z_0 \subseteq B_0^*$ and $Z_1 \subseteq B_1^*$.

Let j be the value in $\{0, 1\}$ such that $\dot{v} \in B_j^*$, and observe that $N(\dot{v}) \cap Z_j \subseteq N(\dot{v}) \cap B_j^* = \emptyset$. Furthermore, we know that $b_1^* \in B_0^*$ and $b_1^* \in B$, but we do not know whether $b_1^* \in B_0$ or $b_1^* \in B_0$; however, there exists a value i in $\{0, 1\}$ such that either $b_1^* \in (B_i \cap B_j^*)$ or $b_1^* \in (B_{1-i} \cap B_{1-j}^*)$. Observe that $Z_j \subseteq B_j^* \cap B_i$ and $Z_{1-j} \subseteq B_{1-j}^* \cap B_{1-i}$.

Finally, let $B' = cc_{\dot{v}}(\{\dot{v}\} \cup (B_i \setminus N(\dot{v})) \cup B_{1-i})$, and consequently we have $X_i = B \setminus B'$. Z is fully contained in B': the only vertices removed from B by X_i are (i) those in $N(\dot{v}) \cap B_i$, but $N(\dot{v}) \cap B_i \cap Z \subseteq N(\dot{v}) \cap B_i \cap Z_j \subseteq N(\dot{v}) \cap Z_j = \emptyset$, and (ii) the vertices not in the connected component of $\dot{v}$ in $G[\{\dot{v}\} \cup (B_i \setminus N(\dot{v})) \cup B_{1-i}]$, but no such vertex can be in Z as $Z \cup \{\dot{v}\}$ is a prefix of the canonical order of B^*, so it induces a connected subgraph.

We thus have that $Z \cup \{\dot{v}\} \subseteq B'$, meaning that $X_i \cap Z = \emptyset$, which proves the claim. Constructively, we can finally observe how the maximal solution $B'' = $comp$(B')$ is the one produces by the algorithm which increases the proximity to B^*, as we have $\{b_1^*, \ldots, b_h^*, b_{h+1}^*\} \subseteq B'' \tilde{\cap} B^*$ and thus $|B'' \tilde{\cap} B^*| \geq |B \tilde{\cap} B^*| + 1$. □

From this, we immediately obtain the correctness of the algorithm.

Theorem 5.4. *A proximity search algorithm (Algorithm 1), using the* NEIGHBORS$(\cdot)$ *function from Definition 5.2 outputs all Maximal Connected Induced Bipartite Subgraphs of a graph G without duplication with $O(nm)$ delay.*

Proof. The correctness follows from Theorem 4.3. The delay is dominated by the cost of the NEIGHBORS(B) function, i.e., calling $O(n)$ times NEIGHBORS(B, v). The cost of the latter is $O(m)$ time to compute $\mathrm{cc}_v(\cdot)$, and $O(m)$ time to compute the COMP$(\cdot)$ function by Lemma 5.9 (delayed to Section 5.4 for compactness). The statement follows. □

5.2 Listing Maximal Induced Bipartite Subgraphs

We can extend our solution to the non-connected case by building one connected component at a time. We obtain the canonical order by Definition 11.6, that is, a BFS order of each component:

Definition 5.5 (canonical order for induced bipartite subgraphs). The *canonical* order of an *induced bipartite subgraph* B is the sequence $b_1, \ldots, b_{|B|}$ obtained by first ordering the connected components of $G[B]$ by incremental order of smallest-id vertex, then ordering each component by a BFS order (given in Definition 5.1) rooted in its smallest-id vertex.

In essence, this corresponds to ordering each connected component as in the connected case (Definition 5.2), and placing earlier components whose smallest-id vertex is smaller. Looking again at Figure 2, and letting B be the subgraph shown in (d) and B^* as that shown in (e), the canonical order of B is $\langle 1, 2, 7, 8, 11, 10\rangle$, that of B^* is $\langle 1, 2, 7, 9, 12, 10\rangle$. By the definition of proximity for canonical reconstruction, we also obtain $B\tilde{\cap}B^* = \{1, 2, 7\}$.

The removables become simpler for this case, as we can simply remove $N(v) \cap B_i$ for $i = 0, 1$. As a result, the neighboring function is essentially the same as the connected case (Definition 5.2), with minor changes as we do not require the connectivity:

Definition 5.6 (neighboring function for maximal induced bipartite subgraphs).

$$\text{NEIGHBORS}(B, v) = \{\text{COMP}(\{v\} \cup (B_i \setminus N(v)) \cup B_{1-i}) \mid i = 0, 1\}$$

We can then proceed to prove correctness and complexity of this case:

Theorem 5.7. *A proximity search algorithm (Algorithm 1), using the* NEIGHBORS$(\cdot)$ *function from Definition 5.6 outputs all maximal induced bipartite subgraphs of a graph G without duplication with $O(n(m + n\alpha(n)))$ delay.*

Proof. Consider the solutions B and B^*. Let $b_1^*, \ldots, b_{|B^*|}^*$ be the canonical ordering of B^* by Definition 5.6, $B\tilde{\cap}B^* = b_1^*, \ldots, b_i^*$, and $u = b_{i+1}^*$ the canonical extender for B, B^*. Let C be the connected component of B^* containing u. Since all the neighbors of u in B^* must be in its same connected component $C_x^{B^*}$, and

the neighbouring function (Definition 5.6) only removes neighbors of u from B, the function may not remove from B any vertex of B^* that is *not* in C. As for vertices in C, $B\tilde{\cap}B^*$ contains a (possibly empty) prefix of its BFS order, which is itself a connected bipartite subgraph in canonical order. By the correctness of Lemma 5.3, for either $B' = \text{COMP}(\mathrm{cc}_v(\{v\} \cup (B_0 \setminus N(v)) \cup B_1))$ or $B' = \text{COMP}(\mathrm{cc}_v(\{v\} \cup (B_1 \setminus N(v)) \cup B_0))$, this prefix is expanded with u, giving us $B'\tilde{\cap}B^* \supseteq (B\tilde{\cap}B^*) \cup \{u\}$ and proving correctness.

As for the delay, we can see that the cost of NEIGHBORS(B) is bounded as for the connected case by $O(n)$ times the cost of NEIGHBORS(B, v), which is in turn bounded by $O(m + n\alpha(n))$ by Lemma 5.9, proving the statement. □

5.3 Maximal Edge Bipartite Subgraphs

Finally, we show how to adapt the above algorithm to Maximal *Edge* Bipartite Subgraphs, where edge-induced subgraphs are denoted by a set of edges, rather than vertices. In the following, given two sets of vertices A and B, let $E(A, B)$ be the set of edges with one endpoint in A and the other in B. We observe that the *Maximal* Edge Bipartite Subgraphs of a connected graph are always connected, otherwise some edge could be added to joint components without creating cycles; by the same logic they span all vertices, and may thus be represented by simply a bipartition $\langle B_0, B_1 \rangle$ of $V(G)$, where the bipartite subgraph corresponds to the edges in $E(B_0, B_1)$. For readability, we use the shorthand $E_B \equiv E(B_0, B_1)$ to refer to the edges of the bipartite subgraph B.

We also observe that the problem is hereditary and allows for a polynomial time computable COMP$(\cdot)$ function. We define the canonical order of a solution B by taking the canonical order $b_1, \ldots, b_{|B|}$ of the *vertices* of $G[B]$ according to Definition 5.1,[7] then taking the edges of B in increasing order of their *latter* vertex in the vertex order, and breaking ties by increasing order of the earlier endpoint. This essentially corresponds to "building" B in a similar fashion as in the induced version, but adding one edge at a time incident to the newly selected vertex. The removables for an edge $e = \{a, b\}$, where $a < b$, are as follows.

- $X_0 = (E_B \setminus N_E(a)) \cup \{e\})$,
- $X_1 = (E_B \setminus N_E(b)) \cup \{e\})$.

The principle behind the neighboring function is different but inspired by the induced case: rather than taking a vertex out of the solution and trying to add it to B_0 or B_1, we take an edge $e = \{a, b\}$ with both endpoints in the same B_i, and try to move the two vertices a and b to opposite sides of the bipartition.

This can be achieved by including the edge e in the solution, and then, to preserve the subgraph being bipartite, removing either $N_E(a)$ or $N_E(b)$ from it. Finally we apply the COMP$(\cdot)$ function to obtain a solution that is maximal.

More formally, recalling $E_B \equiv E(B_0, B_1)$, we define NEIGHBORS(B) as

$$\bigcup_{e=\{a,b\}\in E(G)\setminus E_B} \{\text{COMP}((E_B \setminus N_E(a))\cup\{e\}), \ \text{COMP}((E_B\setminus N_E(b))\cup\{e\})\}$$

---

[7] Note that the vertices of $G[B]$ are all of $V(G)$, but to compute the canonical order we need to consider only the edges in the bipartite subgraph $G[B]$.

Consider two solution B and B^*, with $e_1, \ldots, e_{|E(B_0^*, B_1^*)|}$ being the canonical order of B^*. Furthermore, let $B \tilde{\cap} B^* = \{e_1, \ldots, e_h\}$ and $\dot{e} = e_{h+1} = \{a, b\}$ the canonical extender, i.e., the first edge in the ordering of B^* which is not in B.

By the definition of the canonical ordering, we have that $\{e_1, \ldots, e_h\}$ is a connected bipartite subgraph, meaning that it allows a unique bipartition $B' = B'_0, B'_1$ of its incident vertices. As $\{e_1, \ldots, e_h\} \cup \{\dot{e}\}$ is also a connected bipartite subgraph, for some $j \in \{0, 1\}$ we must have both $N(a) \cap B'_j = \emptyset$ and $N(b) \cap B'_{1-j} = \emptyset$.

Since B' is included in B, we must have either (i) $B'_j \subseteq B_i$ and $B'_{1-j} \subseteq B_{1-i}$ or (ii) $B'_{1-j} \subseteq B_i$ and $B'_j \subseteq B_{1-i}$. Recall now that both a and b are assumed wlog to be in B_i, meaning that $N(a) \cap B_i = N(b) \cap B_i = \emptyset$. In the (i) case, we have $N(b) \cap B_{1-i} \cap B'_{1-j} = \emptyset$, so removing $N_E(b)$ from B may not remove any edge of B'. Thus $\text{COMP}((E_B \setminus N_E(b)) \cup \{\dot{e}\})$, which belongs to $\text{NEIGHBORS}(B)$, will contain $(B \tilde{\cap} B^*) \cup \{e\}$.

In the (ii) case, we have $N(a) \cap B_{1-i} \cap B'_{1-j} = \emptyset$, removing $N_E(a)$ may not remove any edge of B'. Thus $\text{COMP}((E_B \setminus N_E(a)) \cup \{\dot{e}\})$, which also belongs to $\text{NEIGHBORS}(B)$, will contain $(B \tilde{\cap} B^*) \cup \{e\}$.

This means that in both cases, $\text{NEIGHBORS}(B)$ will yield a solution S' that contains $(B \tilde{\cap} B^*) \cup \{e\}$, i.e., such that $|S' \tilde{\cap} S^*| > |S \tilde{\cap} S^*|$.

As the complexity is bounded by $O(m)$ calls to the $\text{NEIGHBORS}(B_i, B_{1-i}, e)$ function, whose cost is again bounded by that of $\text{COMP}(\cdot)$, that is $O(m^2)$ time (By Lemma 5.9), the following theorem holds:

THEOREM 5.8. *Maximal (edge-induced) Bipartite Subgraphs can be listed in $O(m^3)$ time delay.*

5.4 Complexity

In order to complete the analysis, let us look at the cost C_t for the three variants considered:

LEMMA 5.9. *C_t is $O(m)$ for Maximal Connected Induced Bipartite Subgraphs, $O(m + n\alpha(n))$ for Maximal Induced Bipartite Subgraphs, and $O(m^2)$ for Maximal Edge-induced Bipartite Subgraphs, where n and m are the number of vertices and edges, and $\alpha(\cdot)$ is the functional inverse of the Ackermann function [44].*

PROOF. C_t is a bound for computing the $\text{COMP}(S)$ function as well as a canonical order. As the latter is computed by a BFS, it takes $O(m)$ time in all three cases, let us then focus on $\text{COMP}(S)$:

Firstly, observe that using $O(n)$ space and standard data structures, we can mark to which bipartition each vertex of S belongs to (using $O(m)$ time to compute the initial bipartition of S), and which vertices have been already tested for addition, in $O(1)$ time per vertex. If a vertex fails the test to be added, it will not be possible to add it later on, so the total cost of $\text{COMP}(S)$ comes from selecting which vertices to test, and testing each of these vertices once.

For the connected case, we must test only vertices adjacent to S (in no particular order): we can find these initial "candidates" in $O(\sum_{x \in S} |N(x)|) = O(m)$ time, marking each vertex as tested the first time so it is not tested again. Whenever trying to add a vertex v to S, we must pay $O(|N(v)|)$ time to check that all its neighbors belong to the same bipartition of S, in which case v belongs to the other one. If v is not addible, we immediately discard it. If instead we add it to S, we mark it with the correct bipartition, and update the list of candidate with its neighbors in $O(|N(v)|)$ time. As each

vertex is only tested once and only added once, the total cost of C_t is bounded by $O(m)$.

For the non-connected case, we further keep track of connected components via union-find [44] (actually, for each connected component we will keep track of its two partitions). To test a vertex v we must just check that it does not connect to two vertices in different partitions C_0 and C_1 of the same connected component C of X: this can be done in $O(|N(v)|)$. Updating the union-find can be done in total $O(n\alpha(n))$, where $\alpha(\cdot)$ is the functional inverse of the Ackermann function [44].[8]

Once we tested a vertex, if this was not addible, it will never become addible, thus we only need to test each vertex once. The cumulative cost for testing will be the sum of the degrees of all tested vertices, that is bounded by $O(m)$. The total time is thus $O(m + n\alpha(n))$.

Finally, for Maximal Edge-induced Bipartite Subgraphs, we need to test each edge for addition just once as the property is hereditary. For each test we can simply check if the resulting graph is bipartite, which takes $O(m)$ time, for a total cost of $O(m^2)$.   □

6 MAXIMAL K-DEGENERATE SUBGRAPHS

We here consider the enumeration of maximal k-degenerate subgraphs, giving an algorithm that has polynomial delay when k is bounded.

A graph G is k-degenerate if it allows an elimination order where each vertex has degree at most k when deleted. Equivalently, it is k-degenerate if no subgraph of G is a $(k+1)$-core, that is a graph where each vertex has degree greater or equal to $k + 1$. The *degeneracy d* of G is the smallest k for which G is k-degenerate.

A *degeneracy ordering* of G is an order of its vertices in which each vertex v has at most d neighbors occurring later than v, where d is the degeneracy of G. It is well known that a degeneracy ordering can be found in $O(m)$ time by iteratively removing the vertex of smallest degree [2]. To remove ambiguity, when multiple vertices have the same degree we can remove the one with smallest label.

The degeneracy is a well-known sparsity measure [20]; its definition generalizes that of independent sets (0-degenerate graphs) and trees and forests (connected and non-connected 1-degenerate graphs). Furthermore, degeneracy is linked to planarity as all planar graphs are 5-degenerate, while outerplanar graphs are 2-degenerate [35].

We are interested in listing all maximal k-degenerate subgraphs of a graph G. An output-polynomial algorithm is known for maximal *induced* k-degenerate subgraphs if G is chordal [14], but no output-polynomial results are known for general graphs.

6.1 Maximal Induced k-Degenerate Subgraphs

A subgraph of a k-degenerate graph is k-degenerate so the property is hereditary, and degeneracy can be computed in linear time so we can implement the $\text{COMP}(\cdot)$ function in polynomial time.

Given a maximal induced k-degenerate subgraph S, we define its *canonical order* as the *reverse* of its degeneracy ordering, i.e., an ordering $s_1, \ldots, s_{|S|}$, such that $s_{|S|}, \ldots, s_1$ is the degeneracy ordering

---

[8] As $\alpha(n)$ grows extremely slowly, we remark that $\alpha(n)$ is in essence $O(1)$ on real, finite, graphs.

of S. In the case of non connected subgraphs, this is adapted by considering the connected components one at a time in lexicographical order. Then, the proximity is defined by Definition 4.1.

In the resulting ordering we have $|N(s_i) \cap \{s_1, \ldots, s_{i-1}\}| \leq k$, i.e., the neighbors of s_i in S that precede s_i in the canonical order are at most k. This is the key property that gives us the intuition for the algorithm: the removables correspond to all neighbors of the canonical extender *except* a set of size at most k. The neighboring function is obtained as follows.

Definition 6.1 (Neighboring function for Maximal Induced k-Degenerate Subgraphs).

$$\text{NEIGHBORS}(S) = \bigcup_{v \in V(G)} \text{NEIGHBORS}(S, v)$$

Where $\text{NEIGHBORS}(S, v) = \{\text{COMP}(\{v\} \cup S \setminus (N(v) \setminus K) : K \subseteq (S \cap N(v))$ and $|K| \leq k\}$

Less formally, when computing $\text{NEIGHBORS}(S, v)$, we try to add v to S as canonical extender. Since S is maximal, this violates the degeneracy constraint, so we remove all neighbors of v except at most k (the *removable* set being $N(v) \setminus K$). The resulting subgraph $D = \{v\} \cup S \setminus (N(v) \setminus K)$ is k-degenerate: as $D \setminus \{v\}$ is k-degenerate as it is a subgraph of S, and any degeneracy ordering of $D \setminus \{v\}$ becomes a k-degenerate ordering for D if we prepend v in the beginning, because v has at most k neighbors in D. This means $N(v) \setminus K$ is a suitable removable according to Definition 4.2.

We now show how these choices for K satisfy Definition 4.2: we iteratively try for K all possible subsets of $S \cap N(v)$ of size at most k. These combinations, i.e., the number of removables, are $O(\sum_{i \in \{1, \ldots, k\}} \binom{|N(v)|}{i}) = O(n^k)$, which is polynomial when k is bounded.

Let us now look at a target solution S^* such that v is the canonical extender for S, S^*, and let $s_1^*, \ldots, s_{|S^*|}^*$ be the canonical order of S^*: if $v = s_i$ in this order, it follows that $S \tilde\cap S^* = \{s_1^*, \ldots, s_{i-1}^*\}$, and $|N(v) \cap \{s_1^*, \ldots, s_{i-1}^*\}| \leq k$.

We also have that $\{s_1^*, \ldots, s_{i-1}^*\} \subseteq S$, so $N(v) \cap \{s_1^*, \ldots, s_{i-1}^*\} \subseteq N(v) \cap S$: since we try as K all possible subsets of $N(v) \cap S$ of size at most k, we will eventually have $K = N(v) \cap \{s_1^*, \ldots, s_{i-1}^*\}$.

At this point the neighboring function will yield $S' = \text{COMP}(\{v\} \cup S \setminus (N(v) \setminus K) = \text{COMP}(\{v\} \cup S \setminus (N(v) \setminus (S \tilde\cap S^*))$. In other words, we only remove some neighbors of v from S, but all the neighbors that are part of $S \tilde\cap S^*$ are not removed, thus $S' \supseteq \{v\} \cup (S \tilde\cap S^*)$, meaning $|S' \tilde\cap S^*| > |S \tilde\cap S^*|$.

As for the running time, let us consider the cost C_t of a $\text{COMP}(X)$ call. k-degenerate graphs are hereditary, i.e., if a vertex is not addible it will not become addible later, so we need to test each $v \in V(G) \setminus X$ for addition at most once. As testing the degeneracy takes $O(m)$ time, $C_t = O(mn)$ time.

Consider now $\text{NEIGHBORS}(S, v)$: firstly, we enumerate each possible $K \subseteq N(v) \cap S$, which takes $O(\sum_{i \in \{1, \ldots, k\}} \binom{|N(v)|}{i}) = O(|N(v)|^k)$ time. For each, we run $\text{COMP}(\{v\} \cup (S \setminus N(v)) \cup K)$, which takes $O(mn)$ time. The total cost is $O(n^{k+1}m)$ time.

The problem is thus proximity searchable, and the delay of the listing algorithm is the cost of $\text{NEIGHBORS}(S)$, i.e., running $O(n)$ times $\text{NEIGHBORS}(S, v)$ (maintaining $\mathcal{S}$ is negligible). More formally:

THEOREM 6.2. *Maximal Induced k-degenerate Subgraphs are proximity searchable when k is constant, and can be enumerated in $O(mn^{k+2})$ time delay.*

We now observe that 1-degenerate subgraphs are exactly forests, and the connected ones are trees; setting $k = 1$ we immediately obtain polynomial-delay algorithms for listing Maximal Induced Forests that could be easily adapted to Maximal Induced Trees. However, an ad-hoc analysis, delayed to Section 12.2, shows we can obtain algorithms with better delay, and even reduce the space usage to polynomial for these problems.

6.2 Maximal Edge-induced k-Degenerate Subgraphs

We now consider Maximal Edge-induced k-Degenerate Subgraphs, i.e., maximal sets of edges $E \subseteq E(G)$ that correspond to a k-degenerate subgraph of G. An algorithm for this case can be obtained by exploiting the structure of the induced one. In the following, let $N_E(v)$ be the *edge neighborhood* of v, i.e., the set of edges of G incident to the vertex v. Note that edge-induced k-degenerate subgraphs are also hereditary, and so $\text{COMP}(\cdot)$ takes polynomial time.

Let S be an edge-induced k-degenerate subgraph, and let $v_1, \ldots, v_l$ be the canonical order of the vertices of $G[S]$ (i.e., the graph containing only edges in S and vertices incident to them), as in Section 6.1.

The canonical ordering of S is obtained by selecting the edges of B by increasing order w.r.t. their *later* endpoint in the vertex order, breaking ties by order of the other (earlier) endpoint.

This corresponds to selecting the vertices $v_1, \ldots, v_l$ in order, and for each adding the edges towards the preceding vertices one by one. Whenever all the edges from v_i to the preceding vertices have been added, we can observe that the graph corresponds to that induced in $G[S]$ by the vertices $\{v_1, \ldots, v_i\}$. By the canonical order of the vertices defined in Section 6.1, this means v_i has at most k neighbors in $\{v_1, \ldots, v_{i-1}\}$.

Again, the proximity $\tilde\cap$ is given by Definition 4.1.

We can now define the neighboring function:

Definition 6.3 (Neighboring function for Maximal Edge-induced k-Degenerate Subgraphs). Let S be a maximal edge-induced k-degenerate subgraph, and $e = \{a, b\}$ an edge not in S. We define:

$$\text{NEIGHBORS}(S) = \bigcup_{e = \{a,b\} \in E \setminus S} \text{NEIGHBORS}(S, a, b) \cup \text{NEIGHBORS}(S, b, a)$$

Where $\text{NEIGHBORS}(S, a, b) = \{\text{COMP}(\{e\} \cup (S \setminus N_E(a)) \cup K) : K \subseteq (S \cap N_E(a))$ and $|K| \leq k - 1\}$

In other words, we add an edge $e = \{a, b\}$ to S, then force a (or, respectively, b) to have degree at most k, by removing all other edges incident to it except at most $k - 1$, as well as adding e. The resulting graph is k-degenerate as a (respectively b) has degree k, and the residual graph is a subgraph of S, which is k-degenerate, so it is possible to compute a degeneracy ordering.

Consider now two solutions S, S^*, with $S \tilde\cap S^* = \{e_1, \ldots, e_h\}$, and let $\dot{e} = \{x, y\}$ be the earliest edge in the canonical order of S^* that is not in S, i.e., e_{h+1}. Assume wlog that x comes before y in the canonical (vertex) ordering of S^*. In this ordering, y has at most k

neighbors preceding it, i.e, $|\{e_1, \ldots, e_h\} \cap N_E(y)| \leq k$. Furthermore, by the same definition, all edges incident to y that precede $\dot{e}$ in the ordering must be between y and another vertex which comes earlier than x, and thus than y, in the ordering, thus they may be at most $k - 1$ (k, including $\dot{e}$ itself, from y to x). Let K' be the set of these edges (not including $\dot{e}$).

When computing NEIGHBORS(S, y, x), we consider all subsets of edges in S incident to y of size at most $k - 1$. By what stated above, at some point we will consider exactly K'. In this case, we will obtain $S' = \text{COMP}(\{\dot{e}\} \cup (S \setminus N_E(y)) \cup K')$. This must contain all edges in $\{e_1, \ldots, e_h\}$, as we only removed edges neighboring y, but all those in $\{e_1, \ldots, e_h\}$ were in K'. Thus we have $\{e_1, \ldots, e_h\} \cup \dot{e} = \{e_1, \ldots, e_h, e_{h+1}\} \subseteq S'$, which implies $|S' \tilde{\cap} S^*| > |S \tilde{\cap} S^*|$. The case in which x comes after y in the ordering is similarly satisfied by NEIGHBORS(S, x, y).

Finally, we only need to show that NEIGHBORS(S) takes polynomial time to compute: indeed this is $O(m)$ times the cost of NEIGHBORS(S, y, x), which in turn has the cost of computing COMP$(\cdot)$ once for each possible considered set K. These latter are $O(\binom{N_E(y)}{k-1})$, and the COMP$(\cdot)$ can be easily implemented in $O(m^2)$ (as above, testing degeneracy takes $O(m)$ time and each edge needs to be considered at most once for addition since the problem is hereditary), for a total cost that is polynomial when k is constant. We can thus state the following:

THEOREM 6.4. *Maximal Edge-induced k-degenerate Subgraphs are proximity searchable when k is constant, and can be enumerated with delay $O(\binom{n}{k-1} m^3)$.*

7 MAXIMAL CHORDAL SUBGRAPHS

7.1 Maximal Induced Chordal Subgraphs

A graph G is chordal if every cycle in G of length greater than 3 has a chord, i.e., an edge between two non-consecutive vertices in the cycle. Chordal graphs have been widely studied, and it is known that several problems which are challenging on general graphs become easier on chordal graphs (see, e.g., [3, 10, 37]). While the problem of finding a largest chordal subgraph has been studied [4], to the best of our knowledge there are no known enumeration results.

We here aim at listing Maximal Induced Chordal Subgraphs of G. The problem is hereditary, and chordality can be tested in $O(m)$ time [39], thus COMP$(\cdot)$ takes $O(mn)$ time.

A (sub)graph is chordal iff it allows a *perfect elimination ordering* $\{v_1, \ldots, v_n\}$ of its vertices, i.e., such that for all i, $N(v_i) \cap \{v_{i+1}, \ldots, v_n\}$ is a clique [14]. We can obtain this by recursively removing simplicial vertices, i.e., vertices whose neighborhood in the residual graph is a clique.[9]

As the neighbors of a simplicial vertex form a clique, we observe that removing a simplicial vertex cannot disconnect the residual graph. It is also known that a chordal graph has $O(n)$ maximal cliques, and a vertex v participates in $O(|N(v)|)$ maximal cliques [14].

We use this to define the canonical order, which is then combined with Definition 4.1 to obtain the proximity function $\tilde{\cap}$.

---
[9]To remove ambiguity, we can remove the lexicographically smallest when multiple simplicial vertices are present.

Definition 7.1 (Canonical Order for Maximal (Connected) Induced Chordal Subgraphs). The canonical order $\{s_1, \ldots, s_{|S|}\}$ of S is the *reverse* of its perfect elimination ordering, i.e., such that $\{s_{|S|}, \ldots, s_1\}$ is the perfect elimination ordering.

This way, the neighbors of v that precede v in the ordering form a clique. Furthermore, when S is a connected subgraph, any prefix $\{s_1, \ldots, s_{j \leq |S|}\}$ of the canonical order induces a connected subgraph, because we can iteratively remove the last vertex, which is always simplicial. This means the canonical order satisfies condition (1) of Definition 4.2, in the case of both Maximal Induced Chordal Subgraphs and Maximal Connected Induced Chordal Subgraphs. The neighboring function is defined as follows.

Definition 7.2 (Neighboring function for Maximal (Connected) Induced Chordal Subgraphs).
We define NEIGHBORS$(S) = \bigcup_{v \in V(G) \setminus S}$ NEIGHBORS(S, v).
For the non connected case we define
NEIGHBORS$(S, v) = \{\text{COMP}(S \cup \{v\} \setminus (N(v) \setminus Q)):$
Q is a maximal clique of $G[S \cup \{v\}]$ containing v$\}$
While for the connected case we define
NEIGHBORS$(S, v) = \{\text{COMP}(\text{cc}_v(S \cup \{v\} \setminus (N(v) \setminus Q))):$
Q is a maximal clique of $G[S \cup \{v\}]$ containing v$\}$

Less formally, we add a vertex v to S, then remove all its neighbors except one maximal clique Q (meaning the *removable* by definition of canonical reconstruction will be $N(v) \setminus Q$). In the connected case, we further remove vertices not in the connected component of v.

We can easily see that $S \cup \{v\} \setminus (N(v) \setminus Q)$ is chordal, by showing a perfect elimination ordering: v itself is simplicial as its neighbors form a clique, and can be removed; we can then complete the perfect elimination order as the remaining vertices form an induced subgraph of S, which is chordal as induced chordal subgraphs are hereditary.

We now need to prove the last condition; let S and S^* be two solutions, $S \tilde{\cap} S^* = \{s_1^*, \ldots, s_h^*\}$ and $\dot{v} = s_{h+1}^*$ the earliest vertex in the canonical order of S^* not in S.

By the canonical order, $N(\dot{v}) \cap (S \tilde{\cap} S^*) = N(\dot{v}) \cap \{s_1^*, \ldots, s_h^*\}$ is a clique. When computing NEIGHBORS$(S, \dot{v})$, as we try all maximal cliques, for some Q we will have $N(\dot{v}) \cap (S \tilde{\cap} S^*) \subseteq Q$. The resulting S' will thus contain all neighbors of $\dot{v}$ in $S \tilde{\cap} S^*$, and thus all of $S \tilde{\cap} S^*$, plus $\dot{v}$, meaning that $|S' \tilde{\cap} S^*| > |S \tilde{\cap} S^*|$, which proves the correctness the NEIGHBORS$(\cdot)$ function.

Finally, NEIGHBORS$(\cdot)$ can indeed be computed in polynomial time. We first need to list all maximal cliques containing v in $G[S \cup \{v\}]$: these correspond exactly to the maximal cliques of $G[(S \cap N(v)) \cup \{v\}]$; as v is adjacent to *all* vertices in $(S \cap N(v))$, we can further say that these correspond exactly to all maximal cliques of $G[(S \cap N(v))]$, to which we then add v (v can clearly be added to any clique of $G[(S \cap N(v))]$ since it is adjacent to all its vertices).

This correspondence is important, because $G[(S \cap N(v))]$ is an induced subgraph of S, and thus a chordal graph.

Recall now that a chordal graph has $O(n)$ cliques and they can be listed in $O(m)$ time (e.g., by computing a perfect elimination ordering [39]): $G[S \cap N(v)]$ has at most $|N(v)|$ vertices and $O(|N(v)|^2)$ edges, so we can list all the maximal cliques of $G[S \cap N(v)]$ –and

thus all maximal cliques of $G[S \cup \{v\}]$ containing $v-$ in $O(|N(v)|^2)$ time, obtaining at most $|N(v)|$ maximal cliques.

The cumulative cost of listing all cliques for each $v \in V(G) \setminus S$ is thus bounded by $O(\sum_{v \in V(G) \setminus S} |N(v)|^2) = O(mn)$ time, and the process yields $O(\sum_{v \in V(G) \setminus S} |N(v)|) = O(m)$ maximal cliques. For each clique Q, we must further compute the corresponding $\textsc{comp}(\cdot)$ call: as the problem is hereditary, again we only need to test each vertex at most once for addition, and a chordality can be tested in $O(m)$ time, the cost C_t of a $\textsc{comp}(\cdot)$ is $O(mn)$ time (which dominates the time for checking membership in $\mathcal{S}$). Furthermore, the same bound applied to the connected case, as we simply need to consider vertices for addition only when they become adjacent to the current solution. Scanning the neighborhoods of the vertices that are added to the solution to find these candidates has an additional cost of $O(m)$ which does not affect the $O(mn)$ bound. The total cost will be $O(mn + m \cdot mn) = O(m^2n)$

We can thus state that:

THEOREM 7.3. *Maximal Induced Chordal Subgraphs and Maximal Connected Induced Chordal Subgraphs are proximity searchable, and can be listed with $O(m^2n)$ time delay.*

7.2 Maximal Edge-induced Chordal Subgraphs

An algorithm for the edge version can be obtained by defining the canonical order for the edge-induced subgraph in the same way as for Bipartite Subgraphs, based on the canonical ordering of the vertices (see Definition 5.5).

In this problem too, note how all Maximal Edge-induced Chordal Subgraphs of a connected graph are connected, as we can always add edges to a non-connected subgraph without creating cycles, so we do not need to separately consider the connected and non-connected case.

We can then devise a neighboring function $\textsc{neighbors}(S, (x, y))$ like the first one in Definition 7.2, where we use an edge (x, y) as canonical extender.

When adding an edge (x, y) to a maximal solution S, we try as Q all maximal cliques containing either x or y in $G[S]$. In any S^* (for which (x, y) is the canonical extender) one between x and y will occur later in the canonical ordering; wlog, let us say y. As the canonical ordering is based on a reversed perfect elimination ordering, the neighbors of y preceding y in the canonical order of S^* form a clique (including x as well). Thus the neighboring funciton will eventually consider a clique Q containing y and all its preceding neighbors, and when this happens the proximity with S^* is extended. The number of neighboring solutions generated this way will be $O(\sum_{(x,y) \in E(G)} |N(x)| + N|(y)|) = O(mn)$.

The only further requirement is a polynomial time $\textsc{comp}(\cdot)$ function which needs to be applied to each neighboring solution: this follows from [26], who prove that edge-induced chordal subgraphs are *sandwich monotone*. In other words, if a edge-induced chordal subgraph $S \subseteq E(G)$ is *not* maximal, then there is always a single edge $e \in E(G) \setminus S$ such that $S \cup \{e\}$ is a chordal subgraph.

This means $\textsc{comp}(\cdot)$ can be computed in a greedy way by testing, up to m times, that any of the $O(m)$ remaining edges in the graph can be added, which takes $O(m)$ time, for a total cost $C_t = O(m^3)$. The total cost follows.

THEOREM 7.4. *Maximal Edge-induced Chordal Subgraphs are proximity searchable, and can be listed with $O(m^4n)$ time delay.*

8 MAXIMAL INDUCED PROPER INTERVAL SUBGRAPHS

Interval graphs are a well-known subclass of chordal graphs, whose vertices can be arranged as intervals on a line such that two vertices are adjacent if and only if their intervals intersect. In this section, we present a polynomial-delay enumeration algorithm for Maximal *Proper* Interval Subgraphs, a subclass of interval subgraphs corresponding to interval graphs where no two intervals properly contain another.

Despite chordal graphs, interval graphs, and proper interval graphs being closely related to each other, it is interesting to observe how the three enumeration algorithms proposed here (chordal subgraphs, proper interval subgraphs) and in [7] (interval subgraphs) differ significantly. Furthermore, an interesting open question would be to determine whether it is possible to enumerate Maximal Interval Subgraphs directly via proximity search, or whether there is an intrinsic difference in what can be achieved with retaliation-free paths.

8.1 Maximal Connected Induced Proper Interval Subgraphs

A *proper* interval graph is an interval graph where, in the interval representation, no interval properly contains another. Equivalently, it can be defined as interval graphs that admit a *unit-length* representation, i.e., where all intervals have length 1 [22]. In this section we will adopt this latter definition, and all interval representations considered will be intended as unit-length.

In the following, we show how to enumerate Maximal (Connected) Proper Interval Subgraphs of a graph G. We show the connected version of the problem, and later remark how to adapt it to the non-connected case.

It is important to observe that every connected proper interval (sub)graph S has two unique interval representation represented by a sequence $v_1, \dots, v_{|S|}$, and its reverse.[10] The *canonical order* $v_1, \dots, v_{|S|}$ of a Maximal Proper Interval Subgraph S is defined as the sequence given by the interval representation of S which has as s_1 the smallest among the two possible values.

A graphical example is given in Figure 4 (a),(b),(d),(e).

In order to generate neighboring solutions, we take each of the two representations, and for each we proceed as follows. Firstly, we try all vertices $v \in V(G) \setminus S$ as canonical extender. Next, we want to identify the right way of inserting v in the interval representation of S. As the interval length is fixed to 1, there are $5|S|$ possibilities: for any other $w \in S$, we can place v ending just before / just after the start of w, exactly overlapping w, or starting just before / just after the end of w. We can observe how these capture all possible ways to insert v, since given any other placement we can slide v in any direction until it is about to gain or lose an overlap, i.e., one of the endpoints of v will approach the endpoint of another interval, which puts us into one of the $5|S|$ cases above.

---

[10]Ambiguity may be caused by identical vertices, i.e., adjacent and with the same sets of neighbors, but it can be resolved by taking the smallest-id vertex first.
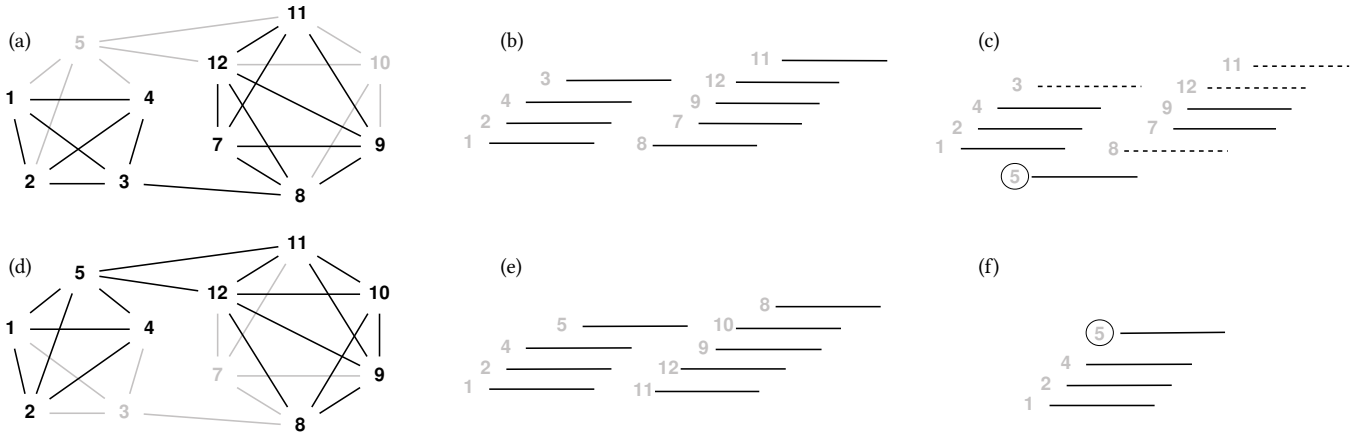
Figure 4: a: a proper interval subgraph S of the graph in Figure 2. d: another proper interval subgraph T. b: the unit interval representation of S, which induces the canonical order 1, 2, 4, 3, 8, 7, 9, 12, 11. e: the representation of T, which induces the canonical order 1, 2, 4, 5, 11, 12, 9, 10, 8.

Finally, we have to make the representation consistent, by removing from $S \cup \{v\}$:

- All vertices (intervals) coming *after* v in this representation.
- All neighbors of v that do *not* overlap its interval.
- All non-neighbors of v that do overlap its interval.
- Finally, all vertices that have become disconnected from the connected component containing v by performing the previous steps.

The resulting representation is consistent with a proper interval graph, so the resulting graph is clearly a proper interval graph. Let us call it X, and let us call $S' = \text{COMP}(\text{cc}_v(X))$. The following example illustrates these operations on the example graphs from Figure 4.

EXAMPLE. *Looking at the graphs in Figure 4 ((a), (d)) and their canonical orderings (see (b),(e)) we can give an example of the operations performed by the neighboring function.*
(c): 5 is added as canonical extender and placed "just before" 3; we remove vertices the dashed vertices 3 and 8 (overlapping 5 but not neighbors of 5 in G), and 11 and 12 (not overlapping 5 but neighbors of 5 in G); 9 and 7 are also removed as not part of the same connected component as 5. The resulting graph (f) is then is maximalized with $\text{COMP}(\cdot)$, and has greater proximity with b than a.
Indeed, $S \tilde{\cap} T = \{1, 2, 3, 4\}$, while the maximal solution obtained maximalizing (f) contains 1, 2, 3, 4, 5.

The set NEIGHBORS(S) will be made of the S' obtained by trying both representations of S, for each all $v \in V(G) \setminus S$, and for each all possible insertions, for a total of $O(n^2)$ neighboring solutions.

We now show that, given S, T, we always obtain some $S' \in$ NEIGHBORS(S) such that $|S' \tilde{\cap} T| > |S \tilde{\cap} T|$.

Let the proximity $S \tilde{\cap} T$ be $t_1, \ldots, t_{i-1}$: this is a prefix of T, and since it is a connected subgraph, its vertices in the same order as in T in one of the two representations of S. Furthermore, let v be our canonical extender.

Looking at the canonical order of T, consider the placement of t_i relative to the preceding intervals $t_1, \ldots, t_{i-1}$. Now, consider the

case where $v = t_i$, the correct one between the two representations of S is considered, and the placement of v relative to $t_1, \ldots, t_{i-1}$ is the same as t_i in T.

As we try all possible placements for $v = t_i$, and as $t_1, \ldots, t_{i-1}$ is a connected subgraph of T, we will also try the placement of t_i considered above. The correct placement of v tells us that, when we remove all intervals coming *after* v from $S \cup \{v\}$, we do not remove vertices from $t_1, \ldots, t_{i-1}$. The placement also tells us that all neighbors of v in $t_1, \ldots, t_{i-1}$ overlap with v in the interval representation, while all non-neighbors of v in $t_1, \ldots, t_{i-1}$ do not, so vertices of $t_1, \ldots, t_{i-1}$ are not removed in the remaining steps. It follows that the set X obtained is a proper interval subgraph of G containing $t_1, \ldots, t_i$, and that $S' = \text{COMP}(X)$ is such that $|S' \tilde{\cap} T| > |S \tilde{\cap} T|$.

8.2 Induced Proper Interval Subgraphs

The non-connected version is similarly solved: as in previous sections, we define the canonical order by ordering each connected component as in the connected case, then ordering the components by smallest-id vertex.

The proximity is then defined by canonical reconstruction (Definition 4.1).

Let $S_1, \ldots, S_j$ be the connected components of S, and $T_1, \ldots, T_k$ those of T, for some T.

We will have that $S \tilde{\cap} T$ consists of some complete connected components of T, as well as a (possibly empty) subset of one component T_i. The canonical extender t_i will be the earliest vertex of the canonical ordering of T_i that is not in S. We can also observe that each of these components is contained itself in some connected component of S.

When we select a canonical extender v, we also select a component S_i (of course, trying all possibilities): we aim to find the component which contains the partially formed T_i.

As we add v to S, we can immediately remove from S all neighbors of v in $S \setminus S_i$, as indeed as t_i only has neighbors in T_i.

We then proceed as in the connected case, ordering S_i in the two possible ways, and trying all $O(n)$ insert possibilities. However,

we must take care of the fact that S_i may contain more than one connected component of T: components *preceding* v in the order are preserved (they cannot overlap v as they precede the other elements of T_i, to which they are not adjacent), but there may be some *following* v.

For this reason, we must introduce another "guess", that is we guess which of the intervals preceding v is t_{i-1} (as, to obtain the correct placement of v in the interval representation, we may not have placed it just after t_{i-1}). Note how the number of possible t_{i-1} is at most $|N(v)|$. This step, which was not necessary in the connected version, allows us to effectively compute a safe way of detaching the intervals following v.

Specifically, we remove from $S_i \cup \{v\}$ the following vertices:

- All vertices (intervals) between t_{i-1} and v.
- All vertices (intervals) coming *after* v in this representation that are adjacent to v or to t_{i-1}.
- All neighbors of v that do *not* overlap its interval.
- All non-neighbors of v that do overlap its interval.

In this way, all the vertices removed could not be part of the proximity $S \tilde{\cap} T$: the ones between t_{i-1} and t_i cannot be in the interval representation since t_{i-1} and t_i are consecutive in the interval representation of T; the others we remove were neighbors of t_{i-1} or t_i, but did not precede them in the interval representation (i.e., in the canonical order), so they could not be part of $S \tilde{\cap} T$. On the other hand, the remaining intervals following v are now in a separate connected component, thus the interval representation of the resulting graph is consistent, contains $t_1, \ldots, t_i$ (for any T, when the correct choices are performed), and we can apply the $\mathrm{COMP}(\cdot)$ function to obtain a neighboring solution S'.

8.3 Running time

As for the complexity, the neighboring function tries $O(n)$ candidates for v, and for each, $5|S| = O(n)$ possible placements in S, in each of the 2 representations. For each, the cost of the procedure is dominated by the application of the $\mathrm{COMP}(X)$ call.

Since we can test whether a graph is a proper interval graph in $O(m)$ time [5], the $\mathrm{COMP}(X)$ function can be implemented in $O(mn)$ time as for chordal subgraphs, giving us a total cost per solution of $O(n^3 m)$ time.

For the non-connected version, we must consider two additional phases: for each v, we firstly selected a connected component S_i of S, and secondly we selected the possible t_{i-1} among the neighbors of v. We can bound the number of connected components by $O(n)$, and, rather than adding another factor $O(n)$ for the choice of t_{i-1}, we can observe how each distinct pair v, t_{i-1} corresponds to an edge, so the number of possible v and t_{i-1} pairs is $O(m)$, for a total complexity of $O(n^3 m^2)$

THEOREM 8.1. *Maximal Induced Proper Interval Subgraphs and Maximal Connected Induced Proper Interval Subgraphs are proximity searchable, and can be listed with $O(n^3 m^2)$ time delay and $O(n^3 m)$ time delay, respectively.*

9 MAXIMAL OBSTACLE-FREE CONVEX HULLS

In application domains such as robotics planning and routing, a common problem is finding areas, typically convex, in a given environment which are free from obstacles (see, e.g., [17, 41]).

In this section we solve the following formulation of the problem: let V and X be two sets of elements, which corresponds to points on a 2-dimensional plane. V represents the point of interest for our application, and X represents the obstacles. For short, let $|V| = j$ and $|X| = h$, and let $n = j + h$ be the total number of points. We are interested in listing all maximal obstacle-free convex hulls (MOCS for short), where an obstacle-free convex hull is a set of elements $S \subseteq V$ such that the convex hull of S does not contain any element of X.

This problem does not concern a graph, but its solutions are modeled as sets of elements, thus the technique may still be applied. Furthermore, we can naturally generalize the problem by adding a graph structure to V, i.e., adding edges between its points, and considering the problem of Maximal *Connected* Obstacle-free Convex hulls.

9.1 Maximal Obstacle-free Convex Hulls

Again, note that the problem is hereditary, i.e., each subset S' of a solution S clearly also admits a convex hull which does not include elements of X (since it will be contained in that of S).

It is worth observing that this is the only problem in this paper to which we do *not* apply the canonical reconstruction strategy.

Consider a maximal solution S and an element $v \in V \setminus S$. As S is maximal, there is at least one element $x \in X$ included in the convex hull of $S \cup \{v\}$. This element x casts two "shadows" S_1 and S_2 on S, seen by v: consider the straight line between v and x, S_1 consists of all elements of S above this line, and S_2 of all those below it. It is straightforward to see how both the convex hull of $S_1 \cup \{v\}$ and that of $S_2 \cup \{v\}$ do not contain x. Any element of S that falls exactly on the line may not participate in any solution involving v.[11] Furthermore, any element $x' \in X$ above this line, and still in the convex hull of $S \cup \{v\}$, further casts two shadows on S_1, as any element below this line casts them on S_2. If we repeat this process for all elements of X in the convex hull of $S \cup \{v\}$ we obtain a number of shadows of S which is at most linear in the number of elements of X. Let $\phi(S, v)$ be the set of these shadows. For each of these shadows $S_i \in \phi(S, v)$, we have that the convex hull of $S_i \cup \{v\}$ may not include elements of X, i.e., $S_i \cup \{v\}$ is a (possibly not maximal) solution.

The neighboring function is then obtained as follows.
Definition 9.1 (Neighboring function for MOCS).

$$\mathrm{NEIGHBORS}(S) = \bigcup_{v \in V(G) \setminus S} \mathrm{NEIGHBORS}(S, v)$$

Where

$$\mathrm{NEIGHBORS}(S, v) = \{\mathrm{COMP}(S_i \cup \{v\}) : S_i \in \phi(S, v)\}$$

Finally, for two solutions S and S^*, we simply define $S \tilde{\cap} S^*$ as the intersection $S \cap S^*$ between their elements.

---

[11]Note that it may not fall between v and x otherwise the convex hull of S would have included x.

Let $I = S \cap S^* = S \tilde{\cap} S^*$, and v any element in $S^* \setminus S$. Since $I \cup \{v\}$ is contained in a MOC, S^*, its convex hull cannot contain any element of X. It follows that I must be fully contained in a single $S_i \in \phi(S, v)$: indeed, if we take two points $u_i \in S_i$ and $u_j \in S_j$, it is evident by the definition of $\phi(S, v)$ that the convex hull of $\{v, u_i, u_j\}$ (or any superset of it) contains at least an element of X. We have that the neighboring function will return $S' = \text{COMP}(S_i \cup \{v\})$, with $I \cup \{v\} \subseteq S'$, which implies $|S' \tilde{\cap} S^*| > |S \tilde{\cap} S^*|$. The algorithm is thus correct.

As for the complexity, the problem is hereditary, so we may compute a $\text{COMP}(S)$ call by testing each vertex in $V \setminus S$ once. The convex hull of S can be computed in $O(|S| \log |S|)$ time [9], and testing a solution consists in checking that each vertex of X is not in this hull, which can trivially be done in $O(|S| \cdot h)$ time. The cost of $\text{COMP}(\cdot)$ is thus $O(j(h + \log j))$ time. For each candidate v, we have at most h neighboring solutions, and since we need to consider at most j candidates, the delay of the algorithm will be $j \cdot h$ times the cost of a $\text{COMP}(\cdot)$ call.

We thus obtain an algorithm with the following complexity:

THEOREM 9.2. *Maximal Obstacle-free Convex Hulls are proximity searchable, and can be listed in $O(j^2 h(h + \log j)) = O(n^4)$ time delay.*

It could be argued that the neighboring function actually reports all solutions of the input-restricted problem in this instance, allowing us to induce a parent-child relationship with the structure of [11, 34], and reducing the space usage to $O(n)$ by using stateless iteration [12]. However, it is worth observing that proximity search required proving a weaker statement, and allows for an arguably simpler proof.

9.2 Maximal Connected Obstacle-free Convex Hulls

We now consider an extension of the problem where on top of V and X we have a graph structure $G = (V, E)$ on the points of V, and we are interested in listing all maximal set of points $S \subseteq V$ such that the convex hull of S is obstacle-free, and $G[S]$ is connected.

We consider this a natural extension as, in the applications mentioned above, it could model requirements on the structure of the obstacle-free areas identified.

The algorithm is remarkably similar to the above version, as the neighboring function still considers $S_i \cup \{v\}$ for all $S_i \in \phi(S, v)$, but only keeps the connected component of $G[S_i \cup \{v\}]$ containing v.

Definition 9.3 (Neighboring function for MOCs).

$$\text{NEIGHBORS}(S) = \bigcup_{v \in V(G) \setminus S} \text{NEIGHBORS}(S, v)$$

Where

$$\text{NEIGHBORS}(S, v) = \{\text{cc}_v(\text{COMP}(S_i \cup \{v\})) : S_i \in \phi(S, v)\}$$

For two solutions S and S^*, we define $S \tilde{\cap} S^*$ as the largest connected component of their intersection $S \cap S^*$. We now prove that there is $S' \in \text{NEIGHBORS}(S)$ such that $|S' \tilde{\cap} S^*| > |S \tilde{\cap} S^*|$.

Let $I = S \cap S^* = S \tilde{\cap} S^*$, and v any element in $S^* \setminus S$ such that $G[I \cup \{v\}]$ is connected. Note that a suitable v must exist, otherwise I would not be connected to the elements of $S^* \setminus I$, contradicting the fact that S^* is a connected solution. Since $I \cup \{v\}$ is contained

in S^*, its convex hull cannot contain any element of X. It follows that I must be fully contained in a single $S_i \in \phi(S, v)$. Furthermore, as $I \cup \{v\}$ is connected, it must be contained in $\text{cc}_v(S_i \cup \{v\})$, the connected component of $G[S_i \cup \{v\}]$ containing v.

Similarly to the above case, we have that the neighboring function will return $S' = \text{COMP}(\text{cc}_v(S_i \cup \{v\}))$, with $I \cup \{v\} \subseteq S'$, which implies $|S' \tilde{\cap} S^*| > |S \tilde{\cap} S^*|$. The algorithm is thus correct.

The complexity can be also derived from the non-connected case: the only additional step is applying $\text{cc}_v(\cdot)$ before the $\text{COMP}(\cdot)$ function. As $\text{cc}_v(\cdot)$ takes $O(m)$ time, where $m = |E(G)| = O(n^2)$, we can conclude the following:

THEOREM 9.4. *Maximal Connected Obstacle-free Convex Hulls are proximity searchable, and can be listed in $O(jh(m + j(h + \log j))) = O(n^4)$ time delay.*

10 MAXIMAL CONNECTED DIRECTED ACYCLIC SUBGRAPHS

In this section we consider a *directed* graph, where each edge has a head and a tail, and its direction is from the tail to the head. We call $N^+(v)$ the *out-neighbors* of the vertex v and $N^-(v)$ its *in-neighbors*.

The goal of this section is listing Maximal Induced Connected Acyclic Subgraphs (MCAIS hereafter) of a given directed graph G. The problem is connected-hereditary, and acyclicity can be tested in $O(m)$ time, thus $\text{COMP}(\cdot)$ can be implemented in $O(mn)$ time.

For completeness, we remark that the non-connected version (Maximal Induced Directed Acyclic Subgraphs), corresponds to listing the complements of Minimal Feedback Vertex Sets in a directed graph, and is of no interest here as an output-polynomial algorithm is given in [42]. We thus address the connected version of the problem, which has no natural counterpart in terms of feedback vertex set. Let us define the canonical order:

Definition 10.1 (Canonical Order for Maximal Connected Induced Acyclic Subgraphs). The canonical order of a MCAIS S is the order $\{s_1, \ldots, s_{|S|}\}$ such that, for each s_i, $\{s_1, \ldots, s_i\}$ is connected, and either $\{s_1, \ldots, s_{i-1}\} \cap N^+(s_i) = \emptyset$ or $\{s_1, \ldots, s_{i-1}\} \cap N^-(s_i) = \emptyset$. If multiple orders are possible let it be the lexicographically minimum.

Our algorithm does not need to compute this order or $\tilde{\cap}$, but we need to show that it always exists.

Firstly, recall that every acyclic graph has at least one source and one target, and let us observe an important property of acyclic graphs with a single source (whose proof trivially follows from the fact that any non-source vertex has a neighbor occurring before itself in the order):

LEMMA 10.2. *Let G be a single-source acyclic connected graph, and $v_1, \ldots, v_n$ any topological order of G. Any prefix $v_1, \ldots, v_i$ of this order induces a connected subgraph.*

Lemma 10.2 also implies that the *reversed* topological order (i.e., where vertices have no forward out-neighbors) of a single-target acyclic connected graph is such that every prefix induces a connected subgraph. We also remark that both these orders satisfy the intersection properties of Definition 10.1.

We now use this lemma to show that the defined canonical order exists for any MCAIS. In the following, we define *collapsing* a set

of vertices $A \subseteq S$ into x as replacing them with a single vertex x, whose in- and out-neighbors correspond to all vertices in $S \setminus A$ that were respectively in- and out-neighbors of some vertex in A.

LEMMA 10.3. *Every Directed Acyclic Graph allows a canonical order by Definition 10.1.*

PROOF. Let S be a Directed Acyclic Graph. Let v_1 be a source of S, and S_1 be the set of vertices reachable by v_1, including v_1. Let $s_{1,1}, \ldots, s_{1,|S_1|}$ a topological ordering of S_1.

No vertex in S_1 can have an out-neighbor outside of S_1 as otherwise said vertex would be in S_1 itself. Let instead S_2 be the set of all vertices in $S \setminus S_1$ that can reach some vertex of S_1.

If we collapse S_1 into a vertex x, we can observe that $S_2 \cup \{x\}$ is acyclic subgraph with x being the only target. Let $x, s_{2,1}, \ldots, s_{2,|S_2|}$ be a reverse topological ordering of $S_2 \cup \{x\}$.

If we replace x with the previously computed order of S_1, we obtain an order $s_{1,1}, \ldots, s_{1,|S_1|}, s_{2,1}, \ldots, s_{2,|S_2|}\}$ which respects Definition 10.1: Each vertex in $s_{1,1}, \ldots, s_{1,|S_1|}$ has no backward out-neighbor by the topological ordering of S_1; each $s_{2,1}, \ldots, s_{2,|S_2|}$ has no backward in-neighbor by the reverse topological ordering of S_2, and because vertices of S_1 can not have out-neighbors outside S_1; finally, every prefix of $s_{1,1}, \ldots, s_{1,|S_1|}, s_{2,1}, \ldots, s_{2,j}$ is connected, as $x, s_{2,1}, \ldots, s_{2,j}$ is connected, meaning that all vertices in $s_{2,1}, \ldots, s_{2,j}$ are connected to some vertex in S_1, that is itself connected.

We may now repeat this step by collapsing $S_1 \cup S_2$ into a vertex x', and since x' will be a source, take S_3 as all vertices reached by x' in $S \setminus (S_1 \cup S_2)$, and take a topological order of $S_3 \cup \{x'\}$, which we append to the order obtained so far (excluding x').

By iterating steps, we obtain an ordering $s_{1,1}, \ldots, s_{1,|S_1|}, s_{2,1}, \ldots, s_{2,|S_2|}, s_{3,1}, \ldots, s_{3,|S_3|} \ldots, s_{k,1}, \ldots, s_{k,|S_k|}$, with $k \leq |S|$, that contains all vertices of S, and such that any prefix will induce a connected subgraph, and any $s_{i,j}$ will have no backward out-neighbors if i is odd, and no backward in-neighbors if i is even, thus there exist an ordering satisfying Definition 10.1 (if a feasible order exists, a lexicographically minimum one must exist too). □

Finally, the proximity $\tilde{\cap}$ follows by Definition 4.1. We define the neighboring function as follows.

Definition 10.4 (Neighboring Function for Maximal Connected Induced Acyclic Subgraphs). For a solution S and a vertex $v \in V(G) \setminus S$, we define

$$\text{NEIGHBORS}(S) = \bigcup_{v \in V(G) \setminus S} \text{NEIGHBORS}(S, v)$$

Where $\text{NEIGHBORS}(S, v) = \{\text{COMP}(\text{cc}_v(\{v\} \cup S \setminus N^+(v))), \text{COMP}(\text{cc}_v(\{v\} \cup S \setminus N^-(v)))\}$

In other words, the function will add v to S. $S \cup \{v\}$ is not acyclic, but all cycles must involve v, so we make it acyclic by removing either all the out-neighbors $N^+(v)$, which makes v a target, or all its in-neighbors $N^-(v)$, which makes v a source. It then takes the connected component containing v and feeds the result to $\text{COMP}(\cdot)$, to surely obtain a MCAIS.

Consider now two solutions S and S^*, and again let $\dot{v}$ be the first vertex in the canonical order of S^* which is not in $S \tilde{\cap} S^*$. More formally, let $S \tilde{\cap} S^* = \{s_1^*, \ldots, s_h^*\}$ and $\dot{v} = s_{h+1}^*$.

Let $S' = \text{COMP}(\text{cc}_{\dot{v}}(\{\dot{v}\} \cup S \setminus N^+(\dot{v})))$ and $S'' = \text{COMP}(\text{cc}_{\dot{v}}(\{\dot{v}\} \cup S \setminus N^-(\dot{v})))$ be the two solutions generated by $\text{NEIGHBORS}(S, \dot{v})$.

By the canonical order of S^*, we have that $(S \tilde{\cap} S^*) \cup \{\dot{v}\}$ is connected, and either $(S \tilde{\cap} S^*) \cap N^+(\dot{v}) = \emptyset$ or $(S \tilde{\cap} S^*) \cap N^-(\dot{v}) = \emptyset$.

It follows that if $(S \tilde{\cap} S^*) \cap N^+(\dot{v}) = \emptyset$, then $(S \tilde{\cap} S^*) \cup \{\dot{v}\} \subseteq \text{cc}_{\dot{v}}(\{\dot{v}\} \cup S \setminus N^+(\dot{v})) \subseteq S'$, and otherwise we have $(S \tilde{\cap} S^*) \cap N^-(\dot{v}) = \emptyset$, which means $(S \tilde{\cap} S^*) \cup \{\dot{v}\} \subseteq \text{cc}_{\dot{v}}(\{\dot{v}\} \cup S \setminus N^-(\dot{v})) \subseteq S''$.

We thus have that either $|S' \tilde{\cap} S^*| > |S \tilde{\cap} S^*|$ or $|S'' \tilde{\cap} S^*| > |S \tilde{\cap} S^*|$, which gives us the second necessary condition of proximity search.

Finally, it is straightforward to see that $\text{NEIGHBORS}(S)$ takes polynomial time, as its cost is bounded by $O(n)$ calls to $\text{COMP}(\cdot)$, which can be implemented in $O(mn)$, meaning that all conditions of Definition 3.1 are satisfied. Theorem 10.5 follows.

THEOREM 10.5. *Maximal Connected Induced Directed Acyclic Subgraphs are proximity searchable, and can be listed $O(mn^2)$ time delay.*

10.1 Maximal Connected Edge-induced Directed Acyclic Subgraphs

We remark here that the structure can be adapted to the edge case, i.e., Maximal Connected Edge-induced Directed Acyclic Subgraphs (MCAES).

As the problem is still hereditary and acyclic subgraphs can be tested in linear time, we can implement the $\text{COMP}(\cdot)$ function in $O(m^2)$ time. The canonical order is as follows.

Definition 10.6 (Canonical order for MCAES). Given a MCAES S, let the canonical ordering of the *vertices* of $G[S]$ according to Definition 10.1 be $v_1, \ldots, v_{|V[S]|}$.

The canonical ordering of S is obtained by selecting the edges of S by increasing order with respect to their *later* endpoint in the vertex order, and breaking ties by increasing order of the other (earlier) endpoint.

We obtain a canonical ordering $e_1, \ldots e_{|S|}$ of S with the following properties: take an edge $e_i = \{v_j, v_k\}$, assuming wlog $j < k$. All edges whose latter endpoint comes earlier than v_k in the vertex order are preceding e_i in the order, thus all edges in the induced subgraph $G[\{v_1, \ldots, v_{k-1}\}]$ will be in the prefix $e_1, \ldots e_i$ of the canonical ordering of S. By Definition 10.1 $G[\{v_1, \ldots, v_{k-1}\}]$ is connected. Finally, the only other edges in $e_1, \ldots, e_i$ are those whose latter endpoint is v_k, so their earlier endpoint is in $\{v_1, \ldots, v_{k-1}\}$. Thus each prefix $e_1, \ldots e_i$ forms a connected (edge) subgraph, which is also acyclic as it is a subgraph of the acyclic subgraph S.

Furthermore, it also holds that, for the latter endpoint v_k of e_i, either

$\{v_1, \ldots, v_{k-1}\} \cap N^+(v_k) = \emptyset$ or $\{v_1, \ldots, v_{k-1}\} \cap N^-(v_k) = \emptyset$. This implies that either $\{e_1, \ldots, e_{i-1}\} \cap N_E^+(v_k) = \emptyset$, or $\{e_1, \ldots, e_{i-1}\} \cap N_E^-(v_k) = \emptyset$, which gives us our neighboring function:

Definition 10.7 (Neighboring Function for MCAES).

Let S be a MCAES and $e = (v_t, v_h)$ a directed edge in $E(G) \setminus S$ directed *from* its tail v_t to its head v_h. Furthermore, let $N_E^+(v_h)$ and $N_E^-(v_t)$ be the out-edges and in-edges of v_h and v_t, respectively. We define $\text{NEIGHBORS}(S, v_t, v_h) = \{\text{COMP}(\text{cc}_{v_t}(\{e\} \cup (S \setminus N_E^-(v_t))), \text{COMP}(\text{cc}_{v_h}(\{e\} \cup (S \setminus N_E^+(v_h))))\}$

And thus

$$\text{NEIGHBORS}(S) = \bigcup_{e = (v_t, v_h) \in E(G) \setminus S} \text{NEIGHBORS}(S, v_t, v_h)$$

In other words, we add e to S, and try each of the two possibilities to obtain the latter vertex in the canonical order of S^*: if it is the tail

v_t of the edge, surely its backward out-neighborhood in the canonical order of S^* is not empty as it contains v_h, so it's in-neighborhood must be, thus we can safely remove $N_E^-(v_t)$ to make $S \cup \{e\}$ acyclic. Conversely, if it is the head v_h we can safely remove $N_E^+(v_h)$. We thus obtain $|S' \widetilde{\cap} S^*| > |S \widetilde{\cap} S^*|$ for some $S' \in \text{NEIGHBORS}(S)$.

We can observe that the cost C_t of a $\text{COMP}(X)$ call is $O(m^2)$ since we can test acyclicity in $O(m)$ time, which we do up to m times, and finding and selecting the edges connected to X take in total $O(m)$ time as well. As the neighboring function produces $O(m)$ solutions, we obtain:

THEOREM 10.8. *Maximal Connected Edge-induced Directed Acyclic Subgraphs are proximity searchable, and can be listed $O(m^3)$ time delay.*

11 PROXIMITY SEARCH IN POLYNOMIAL SPACE

Proximity search consists in a graph traversal, where the number of nodes corresponds to that of solutions. If we store the set of visited nodes, as done in the algorithms presented until now, it follows that the space requirement of the algorithm becomes exponential in n.

Techniques such as reverse-search are able to turn this graph into a rooted tree, that can be traversed without keeping track of visited nodes, by means of a parent-child relationship among solutions, thus achieving polynomial space. However, known instances of reverse search have de facto relied on the problem at hand being hereditary, and the input-restricted problem being solvable in polynomial time (respectively, polynomial total time) to obtain polynomial delay (polynomial total time). Recently, a generalization of reverse-search to non-hereditary properties has been proposed in [13]: this allows us to induce a parent-child relationship for maximal solutions in any *commutable* set system (a class of set systems which includes both hereditary and connected-hereditary properties), and obtain maximal listing algorithm with polynomial space, and whose delay is linked to the input-restricted problem.

In this section we show that, when suitable conditions are met, it is possible to get the best of both worlds: on one hand, using proximity search to overcome the burden of the input-restricted problem and achieve polynomial delay; on the other, using [13] to induce a parent-child relationship among solutions and achieve polynomial space at the same time.

The final goal of the section is proving the following result.

THEOREM 11.1. *Let $(\mathcal{U}, \mathcal{F})$ be a commutable set system, and $\text{NEIGHBORS}(S, s)$ a canonical reconstruction function for a proximity search algorithm (see Definition 4.2). If the canonical order relative to the function $\text{NEIGHBORS}(S, s)$ satisfies the properties of a prefix-closed order (Definition 11.2), the maximal solutions of $(\mathcal{U}, \mathcal{F})$ can be enumerated without duplication in polynomial delay and polynomial space.*

11.1 Requirements and notation of [13]

Let us briefly recall the requirements of [13]. In a set system $(\mathcal{U}, \mathcal{F})$, $\mathcal{U}$ is the *ground set*, i.e., the elements constituting the solutions, and $\mathcal{F}$ defines the solutions, i.e., $S \in \mathcal{F}$ iff $S \subseteq \mathcal{U}$ satisfies the property at hand.

A set system is *strongly accessible* if for any two distinct solutions $S, S' \in \mathcal{F}$ with $S \subset S'$, there exists an element $x \in S' \setminus S$ such that $S \cup \{x\} \in \mathcal{F}$. This is equivalent to saying that any non-maximal solution can be extended into a larger solution with a single element.

We say that a set system is *commutable* if (i) it is strongly accessible, and (ii) it respects the *commutable property*: for any $S, T \in \mathcal{F}$ with $S \subset T$, and any $a, b \in T \setminus S$, we have that $S \cup \{a\} \in \mathcal{F} \wedge S \cup \{b\} \in \mathcal{F}$ implies $S \cup \{a, b\} \in \mathcal{F}$. As mentioned in [13], it is straightforward to see that both hereditary and connected-hereditary properties correspond to commutable set systems.

Furthermore, we call Z the set of "singleton solutions", i.e., $Z = \{e \in \mathcal{U} : \{e\} \in \mathcal{F}\}$, and recall that in any strongly accessible set system $Z \cap S \neq \emptyset$ for any $S \in \mathcal{F}$. We also define, $S^+ = \{x : S \cup \{x\} \in \mathcal{F}\}$.

Given any commutable set system, we can obtain a maximal listing algorithm with two components. Firstly we need an efficient algorithm for solving the input-restricted problem. Secondly, to induce a parent-child structure we need what is called a *family of prefix-closed orders* for the problem, satisfying the following properties:

Definition 11.2 (Prefix-closed orders, from [13]). Let $\Pi(X, v)$ be a family of orders parameterized by $X \in \mathcal{F}$ and $v \in X \cap Z$ such that $\Pi(X, v)$ yields a permutation of $X \cup X^+$. For $X \in \mathcal{F}$ and $v \in X \cap Z$, let us denote by $x_1^v, \ldots, x_k^v$ the elements of X ordered according to $\Pi(X, v)$.[12] We call the family Π *prefix-closed* if for all $X \in \mathcal{F}$ and $v \in X \cap Z$, and $i \in \{1, \ldots, k-1\}$, the following properties hold:

(first) The minimal element is v, i.e., $x_1^v = v$.
(prefix) The i-th prefix $X_i = \{x_1^v, \ldots, x_i^v\}$ of X is a solution, i.e., $X_i \in \mathcal{F}$.
(greedy) The element x_{i+1} is the minimal element of $X_i^+ \cap X$ with respect to the order $\Pi(X_i, v)$.

As explained in [13], each subset X of a maximal solution S does not necessarily belong to $\mathcal{F}$ (as the set system is not necessarily hereditary). The *first* property indicates that we can build S starting from an element $v \in S \cap Z$, whereas the *greedy* property indicates that we can iteratively expand $X = \{v\}$ by considering the elements of $X \cup X^+$ in a prefix-closed order, so that at any point, the prefix $\{x_1, \ldots, x_j\}$ found so far is a solution thank to the *prefix* property.

We use the shorthand notation $\prec_S^t$ to represent $\Pi(S, t)$, where $a \prec_S^t b$ for any two elements $a, b \in \mathcal{U}$ means that a occurs before b in $\Pi(S, t)$.

Given a solution $S \in \mathcal{F}$ we define its *seed*, $\text{SEED}(S)$, as the element of smallest id in $S \cap Z$, i.e., the element s of smallest id in S such that $\{s\} \in \mathcal{F}$. Observe that every non-empty solution S of a strongly accessible set system has a seed: since $\emptyset \subset S$, there is some $s \in S \setminus \emptyset$ such that $\emptyset \cup \{s\} = \{s\} \in \mathcal{F}$.

The simplified notations $\prec_S$ corresponds to $\prec_S^t$ with $t = \text{SEED}(S)$. When S is a maximal solution, $\prec_S$ defines an order $s_1, \ldots, s_{|S|}$ which is called the *solution order* of S.

As in [13], we will also require a *lexicographic* $\text{COMP}(\cdot)$ function: for a solution S, $\text{COMP}(S)$ must be obtained by iteratively adding to S the smallest element in S^+ according to the order $\prec_S$ (i.e., the earliest in $\Pi(S, \text{SEED}(S))$), until S^+ is empty. The resulting solution

---

[12]Note that $x_1 = v$ and that $x_i, x_{i+1} \in X$ are not necessarily consecutive in $\Pi(X, v)$ as some elements from X^+ can be interleaved with them.

is maximal by definition of strongly accessible set systems. We remark that this alternative definition of COMP(S) still returns a maximal solution containing S, and is thus compatible with canonical reconstruction (Definition 4.2).

Finally, given the canonical ordering $s_1, \ldots, s_{|S|}$ of S, the *core* CORE(S) of S is the *longest* prefix $s_1, \ldots, s_i$ of this order such that COMP($s_1, \ldots, s_i$) $\neq S$; its *parent* is PARENT(S) = COMP(CORE(S)) = COMP($s_1, \ldots, s_i$); its parent index is PI(S) = s_{i+1}, i.e., the element following the last one of the core. It follows by definition of parent that COMP(CORE(S) $\cup$ {PI(S)}) = COMP($s_1, \ldots, s_{i+1}$) = S.

The function PARENT(S) defines a forest among solutions, as every solution has a unique parent, except for the ones such that COMP(SEED(S)) = S which are called *roots*, and indeed correspond to the roots of the forest: these are linear in number (as each has a unique seed) and can be found by calling COMP({u}) for any $u \in \mathcal{U}$. The function CHILDREN(P, w) lets us perform a traversal of this structure, since it will find all S such that P = PARENT(S) and w = PI(S).

11.2 Combining proximity search with [13]

Algorithm 2: Polynomial-space proximity search

Input : Commutable set system ($\mathcal{U}, \mathcal{F}$)
      Prefix-closed order family $\preceq_S^s$
      NEIGHBORS(S, s) for canonical reconstruction based on $\preceq_S^s$
Output: All maximal $X \in \mathcal{F}$

1 **foreach** S such that COMP(SEED(S)) = S **do**
2   ENUM(S)

  Function ENUM(X)
    /* Output X if depth is odd   */
3   **foreach** $w \in \mathcal{U} \setminus X$ **do**
4   **foreach** $S \in$ CHILDREN(X, w) **do**
5   ENUM(S)

    /* Output X if depth is even   */

6 **Function** CHILDREN(P, w)
7   **foreach** $R \in$ NEIGHBORS(P, w) **do**
8   **foreach** $s \in (R \cap Z) \setminus \{w\}$ **do**
9           $prefix \leftarrow \{x \in R : x \preceq_R^s w\}$;
10          $S \leftarrow$ COMP($prefix$);
11   **if** $\langle$PARENT(S), PI(S), SEED(S), R(S)$\rangle = \langle P, w, s, R \rangle$
            then yield S;

12 **Function** R(S)   /* finds the first R that can generate S */
13     $P \leftarrow$ PARENT(S);
14     $w \leftarrow$ PI(S);
15     $s \leftarrow$ SEED(S);
16   **foreach** $R \in$ NEIGHBORS(P, w) **do**
17        $prefix \leftarrow \{x \in R : x \preceq_R^s w\}$;
18   **if** COMP($prefix$) = S **then return** R;

When using proximity search in the *canonical reconstruction* flavour, we use a canonical order to define the proximity by Definition 4.1, and a suitable NEIGHBORS(S, s) function such that together

they satisfy Definition 3.1. In this section we show that we can combine proximity search and [13] for commutable properties, if we can produce a canonical order for the canonical reconstruction that corresponds to the solution order induced by $\prec_S$.

We then show in Section 11.3 that it is possible to meet these conditions for canonical orderings that are defined in a greedy way, e.g., by a BFS order like in bipartite subgraphs. Assuming that we meet these conditions, i.e., we have a NEIGHBORS(S, s) function that fits canonical reconstruction (Definition 4.2), based on a canonical order defined by a prefix-closed order $\preceq_S$, we define a variant of [13], showed in Algorithm 2.

The main idea behind this combination comes from the following observation: the parent P = PARENT(S) = COMP(CORE(S)) of S is obtained from a prefix of S, and extending this prefix with PI(S), then applying COMP($\cdot$), gives us COMP(CORE(S) $\cup$ PI(S)) = S (see definitions in Section 11.1). On the other hand, we will show that applying Definition 4.1, $P \tilde{\cap} S$ is exactly CORE(S). Relying on the neighboring function NEIGHBORS(P, PI(S)) of canonical reconstruction, and the *core property* defined in [13], we are able to find the set CORE(S) $\cup$ PI(S), and finally obtain S.

We can now state:

THEOREM 11.3. *Given a commutable set system ($\mathcal{U}, \mathcal{F}$), a prefix-closed order family $\preceq_S^s$ for ($\mathcal{U}, \mathcal{F}$), and a function NEIGHBORS(S, s) for canonical reconstruction (Definition 4.2) based on $\preceq_S^s$, Algorithm 2 enumerates all maximal solutions of ($\mathcal{U}, \mathcal{F}$) without duplication in polynomial delay.*

PROOF. To prove the correctness, we show that any S is found in CHILDREN(P, w) when P = PARENT(S) and w = PI(S).

We will first prove that there exists a solution $R \in$ NEIGHBORS(P, PI(S)) (on Line 7) such that CORE(S) $\cup$ {PI(S)} $\subseteq R$.

Consider the proximity $P \tilde{\cap} S$ by Definition 4.1: the longer prefix of the solution order of S that is completely in P must include CORE(S) since P = COMP(CORE(S)). If $w \in P$ then NEIGHBORS(S, w) returns P by Definition 4.2, and indeed $P \supseteq$ CORE(S) $\cup$ {w}.

Otherwise, P does not include PI(S), meaning that $P \tilde{\cap} S$ = CORE(S) and that w is the canonical extender for P, S. Using the neighboring function NEIGHBORS(P, PI(S)) we obtain at least one solution $R \supseteq$ CORE(S) $\cup$ PI(S).

Using the *core property* defined in [13], we are able to use R to retrieve S: It is proven that Lines 7-11 will find and output any solution S such that CORE(S) $\cup$ {PI(S)} $\subseteq R$, a condition which is guaranteed by what stated above.

The **if** on Line 11 removes duplication: any S is found only once out of all invocations of CHILDREN(P, w): when P = PARENT(S), w = PI(S), s = SEED(S), and R = R(S). The function R(S) simply aims at defining deterministically one single $R \supseteq$ CORE(S) $\cup$ {PI(S)} once the other 3 variables have been fixed. It thus follows that this check is passed exactly once out of the whole execution of the algorithm for any solution (other than the roots, found on Line 1).

Line 1 shows that, by definition, all the roots of the forest are explored by Algorithm 2. We just proved that Line 4 discovers all the children of each visited node exactly once, which concludes the proof of the fact that Algorithm 2 visits every maximal solution of ($\mathcal{U}, \mathcal{F}$) without duplication. □

It is also straightforward to see that each recursive call uses polynomial space, and no solution dictionary $\mathcal{S}$ is maintained. However, the depth of the recursion tree is a factor in the space complexity too: to obtain a polynomial space guarantee, we further need to turn the recursive algorithm into a *stateless* iterative one, as has been done in [13].

We can give a general bound with the following parameters: let q be the maximum size of a solution; $\mathcal{R}_T$ be the time required to solve NEIGHBORS(P, w); $\mathcal{R}_N$ a bound on the number of solutions returned by it; C_t be the time required to compute COMP(X) and O_T the time required to compute the canonical order of $X \cup X^+$. As these bounds are all assumed to be polynomial, we observe their space requirements will be polynomial as well.

Thanks to the alternative output technique, the delay will be bounded by the cost of one iteration of ENUM(X), that is, $O(|\mathcal{U}|)$ times the cost of CHILDREN(P, w). In turn, the cost of CHILDREN(P, w) is that of NEIGHBORS(P, w), plus for each of the $O(\mathcal{R}_N)$ solutions R returned, the cost of processing Lines 8-11. [13] proved that this can be done in $O(q(O_T + C_t))$ time for the given definition of R(S). However, our definition of R(S) is different from the one in [13], and has a cost of $O(\mathcal{R}_T + \mathcal{R}_N O_T)$ instead of $O(O_T + C_t)$. Thus, the total cost of processing Lines 8-11 is $O(q(\mathcal{R}_T + \mathcal{R}_N O_T + C_t))$.

We can thus claim the following:

THEOREM 11.4. *Given a commutable set system $(\mathcal{U}, \mathcal{F})$, a prefix-closed order family $\preceq^s_S$ for $(\mathcal{U}, \mathcal{F})$, and a function NEIGHBORS(S, s) for canonical reconstruction (Definition 4.2) based on $\preceq^s_S$, the maximal solutions of $(\mathcal{U}, \mathcal{F})$ can be enumerated in $O(|\mathcal{U}|\mathcal{R}_T + |\mathcal{U}|\mathcal{R}_N q(\mathcal{R}_T + \mathcal{R}_N O_T + C_t))$ time delay and polynomial space.*

11.3 BFS-based canonical reconstruction

In this section, we provide a technique to implement the result of Section 11.2 (Theorem 11.4), i.e., a canonical reconstruction order that matches the prefix-closed order requirements, and can be applied to hereditary and connected-hereditary properties. We call this technique *BFS-based canonical reconstruction*.

While it is possibly not the only way to obtain a suitable order, it is worth defining formally as we will apply it to several problems in the following sections.

We will first define the order for connected-hereditary property, then exploit it to cover the hereditary case.[13]

Definition 11.5 (canonical-BFS order for connected-hereditary properties). Let S be a solution of a connected-hereditary set system, and v any element in S. The canonical order $\Pi(S, v) = s_1, \ldots, s_{|S \cup S^+|}$ is the lexicographical order of the tuples $\langle d_v(s_i), s_i \rangle$, where $d_v(s_i)$ is the distance between s_i and v in $G[S \cup \{s_i\}]$.

In other words, we order nodes first by $d_v(s_i)$, i.e., their distance from v in $G[S]$, and break ties by vertex id. The same logic applies to nodes x of S^+, for which we use the distance from v in $G[S \cup \{x\}]$. This defines $\preceq^s_S$.

EXAMPLE. *For the Maximal Connected Induced Bipartite Subgraph in Figure 2 (b), the order 2, 3, 5, 8, 11, 7, 10 (as defined in Section 4) is given by the tuples $\langle 0, 2 \rangle, \langle 1, 3 \rangle, \langle 1, 5 \rangle, \langle 2, 8 \rangle, \langle 2, 11 \rangle, \langle 3, 7 \rangle, \langle 3, 10 \rangle$.*

---

[13]Notably, this implies that a BFS-based canonical reconstruction algorithm for the non-connected case immediately follows from one for the connected case.

We can observe how this canonical-BFS order $\Pi(S, v)$ satisfies the properties of Definition 11.2:[14]

(first) The first element s_1^v of $\Pi(S, v)$ is v, as $d_v(v) = 0$ and $d_v(\cdot) \geq 1$ for any other vertex.

(prefix) Any prefix $S_i = \{s_1^v, \ldots, s_i^v\}$ of $\Pi(S, v)$ is connected (thus a solution), since for any s_i, the vertices on a shortest path in $G[S]$ to s_1 are at a smaller distance from s_1 and thus occur before s_i.

(greedy) For any $z \in S_i^+ \cap S$, let k be the distance between v and z in $G[S_i \cup \{z\}]$. Since $z \in S_i^+$, there must be some $w \in S_i \cap N(z)$ at distance $k-1$ from v (in $G[S_i \cup \{z\}]$). This means that the distance between v and z in $G[S]$ is still k: otherwise, there would be a vertex $y \in S \setminus S_i$, i.e., after w in the canonical order, that is a neighbor of z and has distance $\leq k-2$ from v in $G[S]$; this leads to contradiction since y would then need to come *before* w in the BFS order.

It follows that the canonical-BFS order is a prefix-closed order. now straightforward to see how this order satisfies the **(first)**, **(greedy)** and **(prefix)** properties of Definition 11.2, and essentially corresponds to the layer order defined in [13].

Definition 11.6 (canonical-BFS order for hereditary properties). Let S be a solution of a hereditary set system, and v any element in S. For each connected component C_i of $G[S]$, we say the *leader* of the component is v if C_i contains v, and otherwise the vertex of smallest id in C_i.

The canonical order $\Pi(S, v) = s_1, \ldots, s_{|S \cup S^+|}$ (defined on $S \cup S^+$) is the lexicographical order of the tuples $\langle cid(S, s_i), d_l(S, s_i), s_i \rangle$, where for s_i in the component C_i, $cid(S, s_i)$ is the id of the leader of C_i, or 0 if this leader is v (assuming wlog 0 is smaller than any other id), and $d_l(S, s_i)$ is the distance from the leader of C_i in $G[C_i]$. Observe how $s_1 = v$. For a vertex x in S^+, we use as $cid(S, x)$ and $d_l(S, x)$ the values obtained in $G[S \cup \{x\}]$.

Less formally, we order each component by a BFS strategy as in the above case (since $G[C_i]$ is connected) using the leader as root (i.e., s if the component contains s, or its smallest id vertex otherwise); then, we concatenate the sequences obtained by putting the one containing s first, followed by the others ordered by id of their leader.

EXAMPLE. *For the Maximal Induced Bipartite Subgraph in Figure 2 (d), the order is 1, 2, 7, 8, 11, 10, given by the tuples $\langle 1, 0, 1 \rangle, \langle 1, 1, 2 \rangle, \langle 7, 0, 7 \rangle, \langle 7, 1, 8 \rangle, \langle 7, 1, 11 \rangle, \langle 7, 2, 10 \rangle$.*

Before proving that this defines a prefix-closed order, let us prove this auxiliary lemma:

LEMMA 11.7. *Let X be a solution and X_i any prefix of its canonical order. The following facts hold:*

- $\forall z \in X_i^+ \cap X, cid(X_i, z) \geq cid(X, z)$
- $cid(X_i, x_{i+1}) = cid(X, x_{i+1})$
- $\forall z \in X_i^+ \cap X, cid(X_i, z) = cid(X, z) \Rightarrow d_l(X_i, z) = d_l(X, z)$.

PROOF. First, the leader of each connected component of X_i is the same as the leader of the corresponding connected component of X (since the leader is always the first element of the connected

---

[14]For completeness, we could equivalently observe that $d_v(s) = LAY^v_S(s)$ according to Definition 7 in [13].

component in a solution order, and prefixes of components are connected as they are in a BFS order).

Moreover, an element z in $X_i^+ \cap X$ is either directly connected to a connected component of X_i, in which case it has the same leader in X_i and in X by what stated above, or it belongs to its own connected component in $G[X_i \cup \{z\}]$, in which case z is its own leader in $G[X_i \cup \{z\}]$, meaning $cid(X_i, z) = z$. Since by definition $cid(X, z) \leq z$, it follows that $cid(X, z) \leq cid(X_i, z)$, proving the first statement.

We now prove that $cid(X_i, x_{i+1}) = cid(X, x_{i+1})$: Either x_{i+1} is directly connected to the last connected component of X_i (in which case we already proved the equality) or it isn't, in which case $cid(X_i, x_{i+1}) = x_{i+1}$. However, in this case x_{i+1} must be its own leader by definition of the order, so it follows that $cid(X, x_{i+1}) = x_{i+1}$, proving the second statement.

Finally, consider $z \in X_i^+ \cap X$ such that $cid(X_i, z) = cid(X, z)$.

If $cid(X, z) = z$ then $d_l(X_i, z) = d_l(X, z) = 0$; otherwise, let x_l be the leader of z in $X_i \cup \{z\}$: z is in the same connected component C_z as x_l in X, and X_i contains a prefix of the canonical-BFS order of C_z; by the properties of the canonical-BFS order, the shortest path from x_l to z is in this prefix, implying the third statement. □

We can now observe how this order for hereditary properties also satisfies the properties of Definition 11.2

(**first**) By definition s_1 is the first element.

(**prefix**) As this order is defined for hereditary properties, it follows that any subset (hence every prefix) is also a solution.

(**greedy**) We proved in Lemma 11.7 that the tuple associated with each element of $X_i^+ \cap X$ with respect to X_i is either the same or lexicographically greater than the tuple with respect to X. As the tuple for x_{i+1} is the same, and since x_{i+1} is the minimum of $X_i^+ \cap X$ with respect to the order in X, it follows that it's also the minimum of $X_i^+ \cap X$ with respect to the order in X_i.

We remark that it is possible to generalize this definition using different functions for $d(\cdot)$ and $d_l(\cdot)$, as long as monotone behaviour can be guaranteed, i.e., $d(X_i, x)$ (resp. $d_l(X_i, x)$) is less than or equal to $d(X, x)$ (resp. $d_l(X, x)$) when X_i is a prefix of X.

12 POLYNOMIAL SPACE ALGORITHMS

In this section we apply the technique defined in Section 11, and give polynomial-space-polynomial-delay proximity search algorithms, proving the bounds given in Theorem 1.2.

For the problems already solved in exponential space in the previous sections, we remark that it is simply necessary to define their canonical order as a canonical-BFS, then apply Theorem 11.4.

12.1 Maximal Bipartite Subgraphs

Looking at the canonical orders defined for Maximal Connected Induced Bipartite Subgraphs (Definition 5.1) and Maximal Induced Bipartite Subgraphs (Definition 5.5), we can see that their definitions match exactly those of canonical-BFS for connected-hereditary and hereditary properties (respectively, Definition 11.5 and Definition 11.6). We can thus immediately apply the polynomial space variant of the algorithm, and we proceed to compute its complexity.

The cost O_T for computing the canonical order will be $O(m)$ in all cases, as it corresponds to performing a BFS, while C_t corresponds to adding edges in a BFS order, which will take $O(m)$ on the connected version, but $O(m+n\alpha(n))$ on the non-connected one due to the need to dynamically maintain the connected components. The neighboring function for both cases produces a constant number of neighboring solutions, meaning $\mathcal{R}_N = O(1)$ and $\mathcal{R}_T = O(C_t)$. At the same time, all operations require no more than $O(m)$ space. Applying Theorem 11.4, we obtain:

THEOREM 12.1. *Maximal Connected Induced Bipartite Subgraphs and Maximal Induced Bipartite Subgraphs of a graph G can be enumerated via BFS-based canonical reconstruction (Algorithm 2) in $O(m)$ space and, respectively, $O(qnm) = O(n^2m)$ and $O(qn(m + n\alpha(n))) = O(n^2(m + n\alpha(n)))$ time delay.*

12.2 Maximal Induced Trees and Forests

As defined above, a forest is an acyclic undirected graph, and a connected forest is called a tree. These are a special cases of k-degenerate subgraphs: 1-degenerate subgraphs are precisely forests, and connected 1-degenerate subgraphs are trees. However, it is worth consider these problems separately, since we can obtain algorithms with lower delay and polynomial space.

It should be observed that listing Maximal Induced Forests corresponds to listing minimal feedback vertex sets in undirected graph: if $S \subset V$ is a Maximal Induced Forest, $V \setminus S$ is a minimal feedback vertex set. A polynomial-delay solution for the enumeration of feedback vertex sets (and thus Maximal Induced Forests) has been proposed in [42]. This result, however, requires exponential space, and does not extend to Maximal Induced Trees.

Furthermore, while the algorithms proposed could be extended to enumerate maximal edge-induced trees and forests, we do not consider it: these correspond to just the spanning trees of a graph, which are already known to be enumerable in polynomial delay and even constant amortized time [43].

Canonical order and neighboring function. Let S be a maximal induced tree.

We define its canonical as a canonical-BFS order (Definition 11.5), i.e., the sequence $s_1, \ldots, s_{|S|}$ given by a BFS order of $G[S]$ rooted in the vertex s_1 of smallest id.

We then define the proximity by canonical reconstruction (Section 4), and we can immediately observe that this order meets the requirements of Section 11. Next, we focus on obtaining a suitable neighboring function.

Definition 12.2 (Neighboring function for Maximal Induced Trees).

We define NEIGHBORS$(S) = \bigcup\limits_{v \in V(G) \setminus S}$ NEIGHBORS(S, v).

Then, NEIGHBORS(S, v) is defined as:

NEIGHBORS$(S, v) = \{\text{COMP}(\text{cc}_v(S \setminus N(v) \cup \{w, v\})) : w \in N(v) \cap S\}$

The key property here is that each vertex $s_i \in S$ has a single neighbor preceding it in the canonical order, corresponding to its parent in the BFS.

Given two solutions S, T, let $t_1, \ldots, t_{|T|}$ be the canonical order of T, and t_i be the canonical extender for S, T, i.e., the vertex for which $S \tilde\cap T = \{t_1, \ldots, t_{i-1}\} \subseteq S$ and $t_i \notin S$. Furthermore, let t_j be

the parent of t_i in the canonical BFS-order of T, observing that $t_j \in \{t_1, \ldots, t_{i-1}\}$.

To find a solution $S' \supseteq \{t_1, \ldots, t_i\}$, we can simply add t_i to S, then remove all neighbors of t_i except t_j so that we have again an acyclic subgraph, and finally discard every vertex not in the same connected component as t_i (which will include $\{t_1, \ldots, t_i\}$). As we do not know which vertex is t_j, we of course try all $O(|N(t_i)|)$ possibilities, thus a suitable S' is always found.

Complexity. Firstly, we can use the NEIGHBORS(S) function to build a proximity search algorithm whose delay is the cost of NEIGHBORS($\cdot$), and whose space is $O(\mathcal{N} \cdot n)$ (where $\mathcal{N}$ is the number of solutions).

We show the cost of NEIGHBORS($\cdot$) -and the delay of the algorithm- to be $O(m^2)$ time: Observe that the cost C_t a COMP(X) call is $O(m)$ time. We first compute the set of vertices adjacent to X, $P = \cup_{x \in X} N(x)$; for each vertex v, we simply need to check that it has exactly one neighbor in X, in $O(|N(v)|)$ time, and discard it otherwise. Whenever we add v vertex to X, we add is neighbors to P again in $O(|N(v)|)$ time. The total cost is $O(\sum_{v \in V(G)} |N(v)|) = O(m)$.

Now consider NEIGHBORS(S, v): for each $w \in N(v)$, we must compute $cc_v(S \setminus N(v) \cup \{w, v\})$, which takes $O(m)$, then apply COMP($\cdot$) which has the same complexity. The cost is thus $O(|N(v)| \cdot m)$. In turn, this means the cost of NEIGHBORS(S) is $O(\sum_{v \in V(G)} |N(v)| \cdot m) = O(m^2)$.

Furthermore, as we are satisfying all conditions of Section 11 (the order defined is a canonical BFS-order and the problem is connected-hereditary), we apply Theorem 11.4 to obtain a BFS-based canonical reconstruction algorithm, with higher delay but polynomial space.

We observe that no component of the algorithm will require more than $O(m)$ space, and their time complexity is as follows: $\mathcal{U} = O(n)$, $\mathcal{R}_T = O(m\Delta)$, $\mathcal{R}_N = O(\Delta)$ (but as observed above, $|\mathcal{U}| \cdot \mathcal{R}_T$ can be better bounded by $O(m^2)$, and $|\mathcal{U}| \cdot \mathcal{R}_N$ can be bounded by $O(m)$), $q = O(n)$, $O_T = O(m)$ and $C_t = O(m)$. The bound of Theorem 11.4 thus resolves to $O(m^2 + mq(m\Delta + \Delta m + m)) = O(mq(m\Delta)) = O(m^2 n^2)$ time. We can thus conclude the following:

THEOREM 12.3. *The Maximal Induced Trees of a graph G can be enumerated in $O(m^2)$-time delay using $O(\mathcal{N}n)$ space, or alternatively in $O(m^2 n^2)$-time delay and $O(m)$ space.*

12.3 Maximal Induced Forests

As showed in Section 11.3, a BFS-based canonical reconstruction algorithm for the non-connected case immediately follows from the connected one.

For completeness, we show how the algorithm for Maximal Induced Forests is obtained:

The canonical order is obtained by Definition 11.6, i.e., a canonical-BFS order of each connected component, where different components are then sorted by their vertex of smallest id.

The neighboring function is essentially obtained from the connected case by removing the use of the $cc(\cdot)$ function (as we do not require solutions to be connected).

Definition 12.4 (Neighboring function for Maximal Induced Forests).

We define NEIGHBORS(S) $= \bigcup_{v \in V(G) \setminus S}$ NEIGHBORS(S, v).

Then, NEIGHBORS(S, v) is defined as:
NEIGHBORS(S, v) $= \{$COMP($S \setminus N(v) \cup w$) $: w \in N(v) \cap S\}$

The complexity of the components of the algorithm is also inherently the same, with the only difference for the cost C_t of the COMP($\cdot$) function: when we add a vertex, we need to make sure that it does not have two neighbors in the same connected component, and update the connected components as we add vertices. The cost of COMP($\cdot$) will thus be $O(m + n\alpha(n))$ time, obtained by the same logic as for Maximal Bipartite Subgraphs (see Section 5.2), while the rest of the operations are exactly as in the connected case, thus bear the same cost.

We can conclude that NEIGHBORS(S, v) takes $O(|N(v)| \cdot (m + n\alpha(n)))$ time, while NEIGHBORS(S) takes $O(\sum_{v \in V(G)} |N(v)| \cdot (m + n\alpha(n))) = O(m(m + n\alpha(n)))$.

Again, we can obtain an exponential-space algorithm using canonical reconstruction proximity search whose delay is the cost of NEIGHBORS(S), and a polynomial-space algorithm using BFS-based canonical reconstruction, whose delay is given by Theorem 11.4.

For the latter, the costs are obtained adapting the connected version with the new cost of COMP($\cdot$): $\mathcal{U} = O(n)$, $\mathcal{R}_T = O((m + n\alpha(n))\Delta)$, $\mathcal{R}_N = O(\Delta)$ (but $|\mathcal{U}| \cdot \mathcal{R}_T$ can be better bounded by $O(m(m+n\alpha(n)))$, and $|\mathcal{U}| \cdot \mathcal{R}_N$ can be bounded by $O(m)$), $q = O(n)$, $O_T = O(m)$ and $C_t = O(m + n\alpha(n))$. The bound of Theorem 11.4 thus resolves to $O(m(m + n\alpha(n)) + mq((m + n\alpha(n)) + \Delta m + (m + n\alpha(n)))) = O(mq(n\alpha(n) + \Delta m))$ time, which we can again upper bound by $O(m^2 n^2)$ time. We can thus conclude the following:

THEOREM 12.5. *The Maximal Induced Forests of a graph G can be enumerated in $O(m^2 n^2)$-time delay and $O(m)$ space.*

13 CONCLUSIONS

We presented proximity search, a technique for the design of efficient enumeration algorithms, based on defining and traversing a solution graph with bounded out-degree. We showed several application cases, considering problems that did not allow efficient algorithms by known methods, and showing that these allow polynomial delay algorithms by proximity search.

We have provided a guideline, called *canonical reconstruction*, aimed at factorizing the most effective ways to apply our technique, and facilitating the design of efficient algorithms.

We have further shown a technique that, under suitable conditions, allows us to design proximity search algorithms that require only polynomial space. The results are polynomial-delay and polynomial-space algorithms for several problems whose input-restricted problem cannot be solved in polynomial time, including non-hereditary ones.

This paper "breaks the barrier" of the input-restricted problem, showing that its complexity does not imply lower bounds in terms of time or space, nor even a trade-off between the two. This closes questions left open since [11], furthering our understanding on the complexity of enumeration in set systems.

At the same time, this reinvigorates the open question of which listing problems allow efficient algorithms and which do not, and to define a more complete theory of enumeration complexity. On top of being a useful tool to design efficient algorithms for specific problems, we hope that this technique will be able to help us gain more insight into this general question.

ACKNOWLEDGEMENTS

REFERENCES

[1] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21 – 46, 1996.
[2] Vladimir Batagelj and Matjaz Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.
[3] Jean R. S. Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, pages 1–29. Springer, 1993.
[4] Ivan Bliznets, Fedor V Fomin, Michał Pilipczuk, and Yngve Villanger. Largest chordal and interval subgraphs faster than 2^n. *Algorithmica*, 76(2):569–594, 2016.
[5] Kellogg S Booth and George S Lueker. Linear algorithms to recognize interval graphs and test for the consecutive ones property. In *Proceedings of the seventh annual ACM symposium on Theory of computing*, pages 255–265, 1975.
[6] Caroline Brosse, Aurélie Lagoutte, Vincent Limouzy, Arnaud Mary, and Lucas Pastor. Efficient enumeration of maximal split subgraphs and sub-cographs and related classes, 2020.
[7] Yixin Cao. Enumerating maximal induced subgraphs. *CoRR*, abs/2004.09885, 2020.
[8] Nofar Carmeli, Batya Kenig, Benny Kimelfeld, and Markus Kröll. Efficiently enumerating minimal triangulations. *Discrete Applied Mathematics*, 2020.
[9] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, Apr 1996.
[10] L. Sunil Chandran. A linear time algorithm for enumerating all the minimum and minimal separators of a chordal graph. In *COCOON 2001*, pages 308–317, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
[11] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *Journal of Computer and System Sciences*, 74(7):1147 – 1159, 2008.
[12] Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. Sublinear-space bounded-delay enumeration for massive network analytics: Maximal cliques. In *ICALP 2016*, pages 148:1–148:15, 2016.
[13] Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. Listing maximal subgraphs satisfying strongly accessible properties. *SIAM J. Discrete Math.*, 33(2):587–613, 2019.
[14] Alessio Conte, Mamadou Moustapha Kanté, Yota Otachi, Takeaki Uno, and Kunihiro Wasa. Efficient enumeration of maximal k-degenerate subgraphs in a chordal graph. In *COCOON 2017*, pages 150–161, 2017.
[15] Alessio Conte, Andrea Marino, Roberto Grossi, Takeaki Uno, and Luca Versari. Proximity search for maximal subgraph enumeration. *arXiv preprint arXiv:1912.13446*, 2019.
[16] 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*, STOC 2019, pages 1179–1190, New York, NY, USA, 2019. ACM.
[17] Robin Deits and Russ Tedrake. *Computing Large Convex Regions of Obstacle-Free Space Through Semidefinite Programming*, pages 109–124. Springer International Publishing, Cham, 2015.
[18] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, 2005.
[19] Khaled Elbassioni, Kazuhisa Makino, and Imran Rauf. Output-sensitive algorithms for enumerating minimal transversals for some geometric hypergraphs. In *European Symposium on Algorithms*, pages 143–154. Springer, 2009.
[20] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithmics*, 18, 2013.
[21] Fedor V. Fomin, Fabrizio Grandoni, Artem V. Pyatkin, and Alexey A. Stepanov. Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. *ACM Transactions on Algorithms*, 5(1):9:1–9:17, December 2008.
[22] Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific journal of mathematics*, 15(3):835–855, 1965.
[23] Alain Gély, Lhouari Nourine, and Bachir Sadi. Enumeration aspects of maximal cliques and bicliques. *Discrete Applied Mathematics*, 157(7):1447 – 1459, 2009.
[24] Petr A. Golovach, Pinar Heggernes, Mamadou Moustapha Kanté, Dieter Kratsch, Sigve H. Sæther, and Yngve Villanger. Output-polynomial enumeration on graphs of bounded (local) linear mim-width. *Algorithmica*, 80(2):714–741, Feb 2018.
[25] Petr A Golovach, Pinar Heggernes, Dieter Kratsch, and Yngve Villanger. An incremental polynomial time algorithm to enumerate all minimal edge dominating sets. *Algorithmica*, 72(3):836–859, 2015.
[26] 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.
[27] David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119 – 123, 1988.
[28] Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM Journal on Discrete Mathematics*, 28(4):1916–1929, 2014.
[29] Donald E Knuth. *The art of computer programming, Volume 3: Sorting and Searching*. Pearson Education India, 2011.
[30] Donald E Knuth. *The art of computer programming, volume 4A: combinatorial algorithms, part 1*. Pearson Education India, 2011.
[31] Ina Koch, Thomas Lengauer, and Egon Wanke. An algorithm for finding maximal common subtopologies in a set of protein structures. *Journal of Computational Biology*, 3(2):289–306, 1996.
[32] Stefan Kratsch and Magnus Wahlström. Compression via matroids: a randomized polynomial kernel for odd cycle transversal. *ACM Transactions on Algorithms*, 10(4):20, 2014.
[33] Kazuhiro Kurita and Yasuaki Kobayashi. Efficient enumerations for minimal multicuts and multiway cuts, 2020.
[34] Eugene L. Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.
[35] Don R Lick and Arthur T White. k-degenerate graphs. *Canadian J. of Mathematics*, 22:1082–1096, 1970.
[36] John W. Moon and Leo Moser. On cliques in graphs. *Israel journal of Mathematics*, 3(1):23–28, 1965.
[37] Yoshio Okamoto, Takeaki Uno, and Ryuhei Uehara. Linear-time counting algorithms for independent sets in chordal graphs. In Dieter Kratsch, editor, *WG 2005*, pages 433–444, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
[38] Marvin C Paull and Stephen H Unger. Minimizing the number of states in incompletely specified sequential switching functions. *IRE Transactions on Electronic Computers*, EC-8(3):356–367, 1959.
[39] Donald J Rose, Robert Endre Tarjan, and George S Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing*, 5(2):266–283, 1976.
[40] Frank Ruskey. Combinatorial generation. *Preliminary working draft. University of Victoria, Victoria, BC, Canada*, 11:20, 2003.
[41] Sergei Savin. An algorithm for generating convex obstacle-free regions based on stereographic projection. In *SIBCON 2017*, pages 1–6. IEEE, 2017.
[42] Benno Schwikowski and Ewald Speckenmeyer. On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics*, 117(1-3):253–265, 2002.
[43] Akiyoshi Shioura, Akihisa Tamura, and Takeaki Uno. An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM Journal on Computing*, 26(3):678–692, 1997.
[44] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, April 1975.
[45] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006.
[46] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977.
[47] Takeaki Uno. Two general methods to reduce delay and change of enumeration algorithms, 2003. NII Technical Report NII-2003-004E, Tokyo, Japan.
[48] Kunihiro Wasa and Takeaki Uno. Efficient enumeration of bipartite subgraphs in graphs. In *COCOON 2018*, pages 454–466. Springer, 2018.

APPENDIX

A MAINTAINING THE SOLUTION SET IN PROXIMITY SEARCH

For completeness, we briefly describe how to efficiently maintain the $\mathcal{S}$ set with well-known data structures. In the following, let $\mathcal{U}$ be the ground set (e.g., $V(G)$ for vertex-induced graph properties, or $E(G)$ for edge-induced graph properties). Let $\mathcal{N} = |\mathcal{S}|$ be the number of solutions in $\mathcal{S}$, and let $s = \max_{S \in \mathcal{S}}(|S|) \leq |\mathcal{U}|$ be the maximum size of a solution. Recall $s \leq n$ for vertex-induced graph properties, and $s \leq m$ for edge-induced graph properties;

What we aim at showing is that the time for maintaining the solution set is negligible in all cases addressed in this paper: recall

that any solution output by the neighboring function is maximalized, i.e., we apply a COMP($\cdot$) function which adds element to it until it is maximal. If we run COMP($\emptyset$), we can expect to add up to s elements, so its worst-case complexity must be $\Omega(s)$ time.

Binary Decision Diagram [30]. We can see it as a binary tree where leafs are all at depth $|\mathcal{U}|$, and each root-to-leaf path defines a subset of $\mathcal{U}$. We will have a space usage of $O(\mathcal{N} \cdot |\mathcal{U}|)$, while the cost for addition or membership test of a solution will be $O(|\mathcal{U}|)$ time.

This is sufficient for the purpose of our paper as we upper bound s by n (or m, for edge-induced subgraphs) in the complexity results, however it is possible to further improve this using a Trie:

Trie [29]. As above, a solution is represented by a root-to-leaf path. We only have nodes corresponding to including elements, so the depth will be $O(s)$, and so the space usage $O(\mathcal{N} \cdot s)$, however a node may have $O(|\mathcal{U}|)$ children. If we keep these children sorted, we can look them up by binary search and have a cost for addition and membership of $O(s \log |\mathcal{U}|)$; on the other hand, we can get constant time lookup using a hash table, and a cost for addition and membership of $O(s)$ time.