Generating All Maximal Induced Subgraphs for Hereditary and Connected-Hereditary Graph Properties

Sara Cohen [1], Benny Kimelfeld [2], Yehoshua Sagiv [2]

The Selim and Rachel Benin School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904, Israel

Abstract

This paper investigates a graph enumeration problem, called the *maximal $\mathcal{P}$-subgraphs problem*, where $\mathcal{P}$ is a hereditary or connected-hereditary graph property. Formally, given a graph G, the maximal $\mathcal{P}$-subgraphs problem is to generate all maximal induced subgraphs of G that satisfy $\mathcal{P}$. This problem differs from the well-known *node-deletion problem*, studied by Yannakakis and Lewis [1–3]. In the maximal $\mathcal{P}$-subgraphs problem, the goal is to produce all (locally) maximal subgraphs of a graph that have property $\mathcal{P}$, whereas in the node-deletion problem, the goal is to find a single (globally) maximum size subgraph with property $\mathcal{P}$. Algorithms are presented that reduce the maximal $\mathcal{P}$-subgraphs problem to an input-restricted version of this problem. These algorithms imply that when attempting to efficiently solve the maximal $\mathcal{P}$-subgraphs problem for a specific $\mathcal{P}$, it is sufficient to solve the restricted case. The main contributions of this paper are characterizations of when the maximal $\mathcal{P}$-subgraphs problem is in a complexity class C (e.g., polynomial delay, total polynomial time).

Key words: enumeration, graph properties, hereditary properties, maximal subgraphs, complexity classes

Email addresses: `sara@cs.huji.ac.il` (Sara Cohen), `bennyk@cs.huji.ac.il` (Benny Kimelfeld), `sagiv@cs.huji.ac.il` (Yehoshua Sagiv).

[1] Sara Cohen was partially supported by The Israel Science Foundation (Grant 1032/05)

[2] Benny Kimelfeld and Yehoshua Sagiv were partially supported by The Israel Science Foundation (Grant 893/05).

1 Introduction

Hereditary graph properties (i.e., graph properties that are closed with respect to induced subgraphs) and connected-hereditary graph properties (i.e., graph properties that are closed with respect to connected induced subgraphs) include many common types of graph properties such as cliques, bipartite graphs, trees and forests. Such properties appear in many contexts, and thus, they have been widely studied.

One seminal result in the context of hereditary properties is that of Yannakakis and Lewis [1–3] who proved that the *node deletion problem* for hereditary graph properties and for connected-hereditary properties is NP-complete. In other words, given a graph G, a hereditary or connected-hereditary graph property $\mathcal{P}$ and an integer n, it is NP-complete to determine whether there exists an induced subgraph of G with n nodes that has property $\mathcal{P}$. The NP-completeness of the node deletion problem has motivated many researchers to present approximations for the node deletion problem, e.g., [4–7]. Approximations for a related problem, called the edge deletion problem, have also been considered recently [8]. The node deletion problem has also been studied in an online setting [9]. Other interesting research problems that have been studied for hereditary properties $\mathcal{P}$ are the speed of growth of $\mathcal{P}$ [10], the sandwich problem [11], and deciding properties of lexicographically first maximal subgraphs [12].

This paper focuses on the following graph enumeration problem, called the *maximal $\mathcal{P}$-subgraphs problem*: Given *(1)* a hereditary or connected-hereditary graph property $\mathcal{P}$ *and (2)* an arbitrary graph G, find *all* maximal induced subgraphs of G that have the property $\mathcal{P}$. (Note that we are interested in generating *maximal graphs* and not *maximum graphs*, i.e., graphs that are locally maximal, but not necessarily globally maximal.) While the algorithms presented in this paper are for enumeration of maximal induced subgraphs, these results are also immediately applicable to hereditary properties over general subset families. Note that such sets and subsets can be modeled as graphs with no edges.

Graph enumeration problems have been studied independently for many different properties. The problem of finding all maximal cliques and independent sets (a special case of the maximal $\mathcal{P}$-subgraphs problem) was studied in [13–15] and the problem of finding all maximal bipartite cliques in a bipartite graph was studied in [16]. Enumerating spanning trees was considered in [17–19], enumerating trees with a given diameter was studied in [20] and enumerating elementary paths was studied in [17,21]. See [22] for a listing of algorithms for combinatorial enumeration problems.

A general technique for enumeration problems using *reverse search* was presented in [23]. In this paper, the enumeration problem is formulated as a problem of reverse search over a duly defined graph. [23] demonstrated their technique for various problems, such as finding all spanning trees in a graph, and finding all connected induced subgraphs of a graph. Techniques for reducing the delay between outputs of an enumeration algorithm were presented in [24]. Our algorithm $\textsc{RecursiveGen}\langle\mathcal{P}\rangle$ incorporates the ideas of the *modified internal output algorithm* [24] to achieve polynomial delay between outputs. Our algorithm $\textsc{RecursiveGen}\langle\mathcal{P}\rangle$ also bears some resemblance to the algorithm of [25] for generating cut conjunctions and bridge avoiding extensions. However, [25] (1) considers enumerating subgraphs (whereas we enumerate induced subgraphs), (2) does not present an algorithm with polynomial delay, (3) uses a different notion of *extending* partial solutions and (4) presents an algorithm in the context of their specific problem (whereas our setting is for general hereditary and connected-hereditary properties).

This paper differs from previous work in that we do not consider specific properties, but instead, deal with the maximal $\mathcal{P}$-subgraphs problem for a general hereditary or connected-hereditary property $\mathcal{P}$. Our strategy is to solve the maximal $\mathcal{P}$-subgraphs problem using a solution to the input-restricted version of this problem. Formally, the input-restricted maximal $\mathcal{P}$-subgraphs problem is: Given *(1)* a graph property $\mathcal{P}$ *and (2)* a graph G which would be in $\mathcal{P}$ if a single vertex was removed from G, find all maximal induced subgraphs of G that have the property $\mathcal{P}$. It often turns out that the input-restricted maximal $\mathcal{P}$-subgraphs problem has a straightforward solution, whereas a solution to the maximal $\mathcal{P}$-subgraphs problem seems more elusive. For example, for the property $\mathcal{P}_{\text{CLIQUE}}$ that contains all cliques, the input-restricted maximal $\mathcal{P}_{\text{CLIQUE}}$-subgraphs problem is trivial, whereas the maximal $\mathcal{P}_{\text{CLIQUE}}$-subgraphs problem requires a more intricate algorithm, e.g. [13–15]. In this paper, we present algorithms that solve the maximal $\mathcal{P}$-subgraphs problem, given an algorithm that solves the input-restricted version of the maximal $\mathcal{P}$-subgraphs problem.

The output for the maximal $\mathcal{P}$-subgraphs problem may be large—it may even be exponential in the size of the input. Clearly, a polynomial time algorithm for maximal $\mathcal{P}$-subgraphs problem is not attainable, since printing the output may require, in itself, exponential time. Several different complexity measures considered in the past have captured the notion of an efficient algorithm when the output may be large. In general, these measures take into consideration both the size of the input and the size of the output. In this paper, we analyze our algorithms in terms of the complexity measures polynomial total time, incremental polynomial time and polynomial delay, presented in [13]. Note that polynomial total time is the largest of these classes. (Every problem that is either solvable in polynomial delay or in incremental polynomial time is also solvable in polynomial total time.) We are also interested in the complexity classes C PSPACE, where C is one of the three above-mentioned complexity classes (e.g., polynomial-total-time PSPACE). These classes contain problems that can be solved in complexity C, while using space that is polynomial in the input.

The main contributions of this paper are characterizations of when the maximal $\mathcal{P}$-subgraphs problem is in a class C, i.e., results of the following type:

Characterization *For all hereditary/connected-hereditary properties $\mathcal{P}$, the maximal $\mathcal{P}$-subgraphs problem is in class C* if and only if *the input-restricted maximal $\mathcal{P}$-subgraphs problem is in class C.*

In particular, we show the above for hereditary and connected-hereditary properties when C is polynomial total time or incremental polynomial time and for hereditary properties when C is polynomial-total-time PSPACE. We also show sufficient conditions for the maximal $\mathcal{P}$-subgraphs problem to be in a complexity class C by proving results of the following type:

Sufficient Condition *For all hereditary/connected-hereditary properties $\mathcal{P}$, the maximal $\mathcal{P}$-subgraphs problem is in C* if *the input-restricted maximal $\mathcal{P}$-subgraphs problem is in class C'.*

A sufficient condition is shown for hereditary and connected-hereditary properties when C is polynomial delay and C' is PTIME, and for hereditary properties when C is polynomial-delay PSPACE and C' is PTIME.

Our characterizations and sufficient condition are theoretically interesting, since the input-restricted maximal $\mathcal{P}$-subgraphs problem seems intuitively to be easier (Section 3). These results are also practically useful, since they are proven for all properties $\mathcal{P}$, and moreover, for many such properties, finding an efficient algorithms for the input-restricted version is much easier.

The maximal $\mathcal{P}$-subgraphs problem has immediate practical applications in the database field. Interestingly, it turns out that many well-known semantics for answering queries in the presence of incomplete information can be modeled as hereditary or connected-hereditary properties, e.g., [26–29].[3] See [30] for a comprehensive analysis of the relationship between semantics for incomplete information and the maximal $\mathcal{P}$-subgraphs problem.

Full disjunctions were introduced in [31] as an extension of the outer-join operator for an arbitrary number of relations. The problem of computing a full dis-

[3] Some semantics considered in the past can be modeled as rooted-hereditary properties, which are a variation of connected-hereditary properties. All our results for connected-hereditary properties carry over almost immediately to rooted-hereditary graph properties.

junction can be modeled as an enumeration problem for connected-hereditary properties. In [32], using a special case of the techniques shown here, a restricted version of the problem of computing a full disjunction is shown to be solvable in polynomial time, and hence, the general problem is solvable in polynomial delay. In [33] the adaptation of the full-disjunction operator to probabilistic databases [34] is studied, namely, the problem of maximally joining a set of probabilistic relations. This problem is a special case of enumerating maximal induced subgraphs with connected hereditary properties. By reasoning about the restricted problem, they prove that their enumeration problem (and also the restricted one) is computationally equivalent to a natural decision problem. Hence, they show that their problem is generally intractable and in incremental polynomial time (or polynomial delay) for several important classes of schemata.

In some sense, our results serve as a double-edged sword. On the one hand, if an efficient method for solving the input-restricted maximal $\mathcal{P}$-subgraphs problem is found, then the general problem can be solved by applying our algorithms. On the other hand, our results imply that solving the input-restricted maximal $\mathcal{P}$-subgraphs problem is not easier, with respect to several complexity classes, than solving the general problem.

2 Graphs and Graph Properties

A *graph* $G = (V, E)$ consists of a finite set of *vertices* V and a set of *edges* $E \subseteq V \times V$. We use $V(G)$ to denote the set of vertices of G. A graph H is an *induced subgraph* of a graph G, written $H \sqsubseteq G$, if H is derived from G by deleting some of the vertices of G (and the edges incident on these vertices). We will also say that G *subsumes* H. We write $H \sqsubset G$ if $H \sqsubseteq G$ and H is not equal to G.

We use $G[\{v_1, \dots, v_n\}]$ to denote the induced subgraph of G that contains exactly the vertices $v_1, \dots, v_n$. If H and H' are induced subgraphs of G and v is a vertex in G, we use $G[H, v]$ and $G[H, H']$ as shorthand notations for $G[V(H) \cup \{v\}]$ and $G[V(H) \cup V(H')]$, respectively.

A *graph property* $\mathcal{P}$ (or simply *property*, for short) is a nonempty and possibly infinite set of graphs. If $G \in \mathcal{P}$, we say that G *satisfies* $\mathcal{P}$. For example, "is a clique" is a graph property that contains all graphs that are cliques. In this paper, we only consider properties $\mathcal{P}$ for which it is possible to verify whether a graph G is in $\mathcal{P}$ in polynomial time. Hence, we assume that there is a polynomial procedure $\textsc{Sat}\langle\mathcal{P}\rangle$ that receives a graph G as input, and returns **true** if $G \in \mathcal{P}$, and **false** otherwise. We use $s_{\mathcal{P}}(n, m)$ to denote the running time of $\textsc{Sat}\langle\mathcal{P}\rangle$ on a graph with n vertices and m edges. Observe that the notation $\langle \mathcal{P} \rangle$ denotes an algorithm that is parameterized by the property $\mathcal{P}$, i.e., that differs for each value of $\mathcal{P}$.

A property $\mathcal{P}$ is *hereditary* if $\mathcal{P}$ is *closed* with respect to induced subgraphs, i.e., whenever $G \in \mathcal{P}$, every induced subgraph of G is also in $\mathcal{P}$. A property $\mathcal{P}$ is *connected hereditary* if *(1)* all the graphs in $\mathcal{P}$ are connected *and (2)* $\mathcal{P}$ is closed with respect to connected induced subgraphs. Many properties are hereditary [35], e.g., "is a clique," "is a bipartite graph" and "is a forest" or connected hereditary, e.g., "is a star," "is a tree" and "is a connected bipartite graph."

3 Problem Definition

In this paper, we focus on two problems: the *maximal $\mathcal{P}$-subgraphs problem* and the *input-restricted maximal $\mathcal{P}$-subgraphs problem.* This section formally defines and compares these two problems.

Let G be a graph and let $\mathcal{P}$ be a property. (The graph G is not necessarily in $\mathcal{P}$.) We say that H is a *$\mathcal{P}$-subgraph* of G if $H \sqsubseteq G$ and $H \in \mathcal{P}$. We say that H is a *maximal $\mathcal{P}$-subgraph* of G if H is a $\mathcal{P}$-subgraph of G and there is no $\mathcal{P}$-subgraph H' of G, such that $H \sqsubset H'$. We use $\mathcal{P}(G)$ to denote the set of maximal $\mathcal{P}$-subgraphs of G.

We now introduce the first problem of interest.

Problem 1 *The* maximal $\mathcal{P}$-subgraphs problem *is: Given a graph G, generate the graphs in the set $\mathcal{P}(G)$.*

Suppose that we want to show that the maximal $\mathcal{P}$-subgraphs problem is efficiently solvable, for a particular property $\mathcal{P}$, with respect to some complexity class. To do this we must devise an algorithm that, when given any graph G, produces $\mathcal{P}(G)$ efficiently. For many properties $\mathcal{P}$, it is difficult to define such an algorithm, since an arbitrary graph G must be considered. Hence, we focus on a restricted version of the maximal $\mathcal{P}$-subgraphs problem. Some algorithms presented in the past for solving the maximal $\mathcal{P}$-subgraphs problem, for particular properties $\mathcal{P}$, e.g., [13], first solved this restricted problem and then applied the solution to the general problem.

Let G be a graph and let $\mathcal{P}$ be a property. We use $G-v$ to denote the induced subgraph of G that contains all the vertices other than v. We say that G *almost satisfies* $\mathcal{P}$ if there is a vertex v in G, such that $G - v \in \mathcal{P}$.

We now present our second problem of interest.

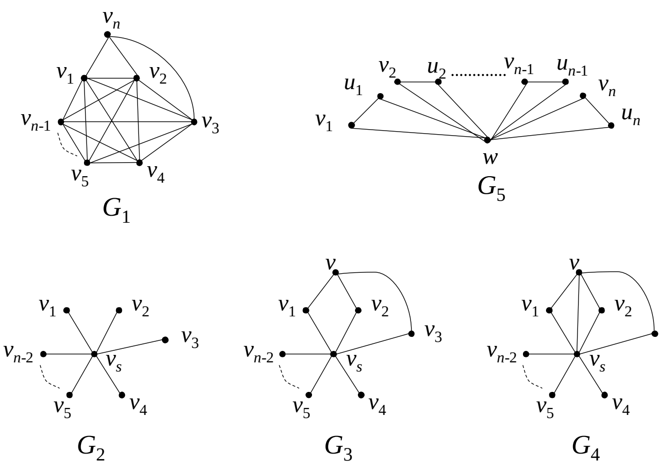
G_1
G_2
G_3
G_4
G_5
v_1 v_2 v_3 v_4 v_5 v_n v_{n-1} v_{n-2} v_s
u_1 u_2 u_{n-1} u_n w

Fig. 1. Graph G_1 almost satisfies $\mathcal{P}_{\text{CLIQUE}}$. Graph G_2 satisfies $\mathcal{P}_{\text{STAR}}$ and graphs G_3, G_4 almost satisfy $\mathcal{P}_{\text{STAR}}$. Graph G_5 almost satisfies $\mathcal{P}_{\text{BIP}}$.

Problem 2 *The* input-restricted maximal $\mathcal{P}$-subgraphs problem *is: Given a graph G that almost satisfies $\mathcal{P}$, generate the graphs in the set $\mathcal{P}(G)$.*

Conceptually, the input-restricted maximal $\mathcal{P}$-subgraphs problem seems to be an easier problem to solve. To support this intuition, we consider two related problems: the node deletion problem and the problem of determining the size of $\mathcal{P}(G)$. We compare the complexity of these problems when applied to arbitrary graphs or to graphs that almost satisfy $\mathcal{P}$.

The node-deletion problem is defined as follows. Given a graph G, a hereditary or connected-hereditary property $\mathcal{P}$ and a number k, determine whether there is an induced subgraph of G that satisfies $\mathcal{P}$ and has k vertices. Obviously, the node-deletion problem is solvable in PTIME if G almost satisfies $\mathcal{P}$. However, this problem is NP-complete for general graphs [3].

Now consider the problem of determining the size of $\mathcal{P}(G)$. This problem is sometimes quite easy if G almost satisfies $\mathcal{P}$. For example, recall the property $\mathcal{P}_{\text{CLIQUE}}$ that contains all cliques. Determining $|\mathcal{P}_{\text{CLIQUE}}(G)|$ is in PTIME if G almost satisfies $\mathcal{P}$ [14]. (We review this result in Example 3.1). However, determining $|\mathcal{P}_{\text{CLIQUE}}(G)|$ is known to be #P-complete [36] for general graphs.

The complexity of the input-restricted maximal $\mathcal{P}$-subgraphs problem is highly dependent on the graph property $\mathcal{P}$. Sometimes it turns out that the input-restricted maximal $\mathcal{P}$-subgraphs problem can actually be solved in polynomial time, since the number of graphs in its output is bounded in size by a constant or by a polynomial function of the input. However, the fact that G almost satisfies $\mathcal{P}$ does not always entail that $\mathcal{P}(G)$ is small. We demonstrate these different situations in the following example.

Example 3.1 We start by considering the property $\mathcal{P}_{\text{CLIQUE}}$. Let G be a graph that almost satisfies $\mathcal{P}_{\text{CLIQUE}}$. We first review the result from [14] that shows that $\mathcal{P}_{\text{CLIQUE}}(G)$ contains at most two graphs. Obviously, if $G \in \mathcal{P}_{\text{CLIQUE}}$, then $\mathcal{P}_{\text{CLIQUE}}(G)$ contains exactly one graph. Otherwise, there is a vertex v in G such that $G - v \in \mathcal{P}_{\text{CLIQUE}}$. Let G' be the induced subgraph of G derived by removing all vertices that are not neighbors of v. One may easily verify that $\mathcal{P}_{\text{CLIQUE}}(G)$ contains exactly the two graphs $G - v$ and G'. As an example, consider the graph G_1 in Figure 1. Observe that G_1 almost satisfies $\mathcal{P}_{\text{CLIQUE}}$, since $G_1 - v_n \in \mathcal{P}_{\text{CLIQUE}}$. The set $\mathcal{P}_{\text{CLIQUE}}(G_1)$ contains the graphs $G_1 - v_n$ and $G_1[\{v_n, v_1, v_2, v_3\}]$.

Now, consider the connected-hereditary property $\mathcal{P}_{\text{STAR}}$ that contains all star graphs. A graph G with n vertices is a star graph if it contains one vertex with degree $n - 1$ and all other vertices have a degree of one. We call the vertex with degree $n - 1$ the *center* of the graph. The graph G_2 in Figure 1 is a star graph with $n - 1$ vertices. Observe that v_s is the center of this star.

Let G be a graph with n vertices that almost satisfies $\mathcal{P}_{\text{STAR}}$. We show that $\mathcal{P}_{\text{STAR}}(G)$ contains at most n graphs. If $G \in \mathcal{P}_{\text{STAR}}$, then $|\mathcal{P}_{\text{STAR}}(G)| = 1$. Suppose otherwise. Let v be a vertex of G such that $G - v \in \mathcal{P}_{\text{STAR}}$. Let v_s be the center of $G - v$. Let $v_1, \ldots, v_k$ be the neighbors of v in G. We distinguish between two cases:

- Suppose that v_s is not among $v_1, \ldots, v_k$. For all $i \leq k$ we define G^i as the graph containing the vertices $\{v_i, v_s, v\}$. Let G' be the graph containing the vertices $\{v, v_1, \ldots, v_k\}$. Intuitively, G^i corresponds to the subgraph of G that has v_i as its center, and G' corresponds to the subgraph of G that has v as its center. Then, $\mathcal{P}_{\text{STAR}}(G) = \{G - v, G^1, \ldots, G^k, G'\}$.
- Suppose that v_s is among $v_1, \ldots, v_k$. Let G^s be the graph containing the vertices $(V(G) - \{v_1, \ldots, v_k\}) \cup \{v_s\}$. Let G' be the graph containing the vertices $\{v, v_1, \ldots, v_k\} - \{v_s\}$. Then, $\mathcal{P}_{\text{STAR}}(G) = \{G - v, G^s, G'\}$. Intuitively, (1) $G - v$ is the graph with v_s as its center, that does not contain v, (2) G_s is the graph with v_s as its center, that also contains v, and (3) G' is the graph with v as its center.

These two cases are depicted with graphs G_3 and G_4 of Figure 1. Observe that at worst, v may have $n - 2$ neighbors, if v_s is not the neighbor of v. It follows from the discussion that $\mathcal{P}_{\text{STAR}}(G)$ contains at most n graphs.

Finally, consider the hereditary property $\mathcal{P}_{\text{BIP}}$ that contains all bipartite graphs. It is possible for the size of $\mathcal{P}_{\text{BIP}}(G)$ to be exponential in the size of G, even if G almost satisfies $\mathcal{P}_{\text{BIP}}$. Consider, for example, the graph G_5 in Figure 1. The graph G_5 almost satisfies $\mathcal{P}_{\text{BIP}}$ since $G_5 - w \in \mathcal{P}_{\text{BIP}}$. However, the set $\mathcal{P}_{\text{BIP}}(G_5)$ contains $2^n + 1$ graphs, i.e., $G_5 - w$ and the graphs derived by choosing the vertex w and one from each pair of vertices (v_i, u_i), for all i. $\square$

Example 3.1 shows that the size of $\mathcal{P}(G)$ varies, even if G almost satisfies $\mathcal{P}$. However, there is a correspondence between the size of $\mathcal{P}(G)$ and the size of $\mathcal{P}(G')$ for induced subgraphs G' of G. This correspondence is presented in the following proposition.

Proposition 3.2 *Let $\mathcal{P}$ be a hereditary property. Let G be a graph with n vertices and let G' be an induced subgraph of G. Then, the following hold:*

(1) if $\mathcal{P}$ is hereditary then each graph in $\mathcal{P}(G)$ subsumes at most one graph in $\mathcal{P}(G')$;
(2) if $\mathcal{P}$ is hereditary, then $|\mathcal{P}(G')| \leq |\mathcal{P}(G)|$.
(3) if $\mathcal{P}$ is connected hereditary then each graph in $\mathcal{P}(G)$ subsumes at most n graphs in $\mathcal{P}(G')$;
(4) if $\mathcal{P}$ is connected hereditary, then $|\mathcal{P}(G')| \leq n\,|\mathcal{P}(G)|$.

Proof. We first show Claim 1. Suppose, by way of contradiction, that Claim 1 does not hold. Then, there is a graph $H \in \mathcal{P}(G)$ for which there are two distinct graphs H_1 and H_2 in $\mathcal{P}(G')$, such that $H_1 \sqsubseteq H$ and $H_2 \sqsubseteq H$. The graph H contains all the vertices of H_1 and all the vertices of H_2 (and perhaps, additional vertices). Let H_3 be the graph $G[H_1, H_2]$. Since $H \in \mathcal{P}$ and $\mathcal{P}$ is hereditary, it follows that $H_3 \in \mathcal{P}$. In addition, H_3 only contains vertices from G'. Hence, H_3 must be subsumed by a graph in $\mathcal{P}(G')$. However, $H_1 \sqsubset H_3$ and $H_2 \sqsubset H_3$, in contradiction to the assumption that H_1 and H_2 are maximal, i.e., that H_1 and H_2 are in $\mathcal{P}(G')$.

We now show Claim 2. First, observe that by Claim 1, each graph in $\mathcal{P}(G)$ subsumes at most one graph in $\mathcal{P}(G')$. The claim follows since each graph H in $\mathcal{P}(G')$ is subsumed by at least one graph H' in $\mathcal{P}(G)$.

We show Claim 3. Suppose, by way of contradiction, that the required does not hold. Then, there is a graph $H \in \mathcal{P}(G)$ for which there are $n+1$ distinct graphs $H_1, \ldots, H_{n+1}$ in $\mathcal{P}(G')$, such that $H_i \sqsubseteq H$, for all $i \leq n+1$. By the Pigeon-Hole Principle, there are two graphs H_i and H_j ($i, j \leq n+1$) such that H_i and H_j share a common vertex v. Consider the graph $G[H_i, H_j]$. This graph is connected, since all the vertices in H_i and in H_j are connected to v. In addition, $G[H_i, H_j] \sqsubseteq H$. Since $\mathcal{P}$ is connected hereditary and $H \in \mathcal{P}$, it follows that $G[H_i, H_j] \in \mathcal{P}$. This contradicts the maximality of H_i and H_j in $\mathcal{P}(G')$.

Finally, Claim 4 follows from Claim 3, since each graph H in $\mathcal{P}(G')$ is subsumed by at least one graph H' in $\mathcal{P}(G)$. $\square$

4 Complexity Classes

This paper explores the problem of computing $\mathcal{P}(G)$, for a hereditary or connected-hereditary property $\mathcal{P}$ and a graph G. The maximal $\mathcal{P}$-subgraphs problem cannot be solved in polynomial time, in the general case. This follows from the fact that the size of $\mathcal{P}(G)$ may be exponential in the size of G. Hence, exponential time may be needed just to print the output.

In [13], several complexity measures were considered to measure the performance of algorithms that may have exponential-size output. We review these complexity measures. In this discussion, we use n to denote the size of the input and K to denote the number of solutions in the output. For the maximal $\mathcal{P}$-subgraphs problem, n is the size of G and K is the number of graphs in $\mathcal{P}(G)$.[4]

Polynomial Total Time. Consider a problem that may, possibly, have many solutions. Such a problem can be solved in *polynomial total time* [13] if the time required to list all its solutions is bounded by a polynomial in n and K. This complexity measure is the same as polynomial time input-output complexity, which is commonly considered in database theory, e.g. [37]. In general, there are enumeration problems not solvable even in this complexity class. For example, generating all maximal 3-colorable subgraphs of a graph cannot be solved in polynomial total time, unless P=NP, since it is NP-complete to determine whether a graph is 3-colorable.

Incremental Polynomial Time. A problem is solvable in *incremental polynomial time* if for all $k \leq K$, it is possible to return k solutions from the output in polynomial time in n and k. (The solutions may be returned in any order.) Observe that every problem that is solvable in incremental polynomial time is also solvable in polynomial total time. Incremental polynomial time is of importance when the user would like to optimize evaluation time for retrieval of k maximal induced subgraphs, as opposed to optimizing for overall time. This is particularly useful in a scenario where the user reads the answers as they are delivered, or is only interested in looking at a small portion of the total result. If a problem is solvable in polynomial total time, but not in incremental polynomial time, the user may have to wait exponential time until the entire output is created, before viewing a single solution, e.g., a single maximal $\mathcal{P}$-subgraph.

Polynomial Delay. A problem is solvable in *polynomial delay* if all the solutions in its output can be generated, one after another, in such a way that the delay before printing the first answer, between any two consecutive solu-

[4] To be more precise, later on we will use both n and m to denote the size of the input G, where n is the number of vertices in G and m is the number of edges in G.

tions, and from the time that the last answer is printed until termination, is bounded by a polynomial in n. Observe that every problem that is solvable in polynomial delay is also solvable in incremental polynomial time. We note in passing that every problem that is solvable in polynomial delay is also in the complexity class P-enumerable [38]. This complexity class contains problems for which solutions can be enumerated in time $K\,p(n)$, where p is a polynomial in n. P-enumerable differs from polynomial total time in that the factor of the output in the running time must be linear.

Polynomial Space. Observe that the maximal $\mathcal{P}$-subgraphs problem is solvable in PSPACE with a naive algorithm that enumerates every subgraph of the input graph and checks if it has the property $\mathcal{P}$ and is maximal. Such an algorithm is clearly not efficient. Thus, PSPACE in itself is not of interest to us. Instead we focus on the classes *polynomial-total-time PSPACE*, *incremental-polynomial PSPACE* and *polynomial-delay PSPACE* which contain problems solvable in polynomial total time, incremental polynomial time and polynomial delay, respectively, while using space that is polynomial in n.

5 The Maximal $\mathcal{P}$-Subgraphs Problem

For hereditary and connected-hereditary properties, we reduce the maximal $\mathcal{P}$-subgraphs problem to the input-restricted maximal $\mathcal{P}$-subgraphs problem. Our reduction is by means of a procedure that shows how to compute $\mathcal{P}(G)$ for an arbitrary graph G, given a procedure that can compute $\mathcal{P}(H)$ for graphs H that almost satisfy $\mathcal{P}$. We present two different procedures for generating $\mathcal{P}(G)$ that fall in different complexity classes, depending on the complexity of the input-restricted maximal $\mathcal{P}$-subgraphs problem.

Both procedures reduce the maximal $\mathcal{P}$-subgraphs problem to the input-restricted version by using the procedure $\textsc{GenRestrHered}\langle\mathcal{P}\rangle(H)$. The procedure $\textsc{GenRestrHered}\langle\mathcal{P}\rangle(H)$ receives a graph H that almost satisfies $\mathcal{P}$ and returns the set $\mathcal{P}(H)$. $\textsc{GenRestrHered}$ is not defined in this paper. Instead, it must be provided on a per-property basis. For instance, Example 3.1 essentially describes the procedure $\textsc{GenRestrHered}\langle\mathcal{P}_{\text{CLIQUE}}\rangle(H)$ and the procedure $\textsc{GenRestrHered}\langle\mathcal{P}_{\text{STAR}}\rangle(H)$, i.e., $\textsc{GenRestrHered}\langle\mathcal{P}\rangle(H)$ for the properties $\mathcal{P}_{\text{CLIQUE}}$ and $\mathcal{P}_{\text{STAR}}$.

Let H be a graph that almost satisfies $\mathcal{P}$. Suppose that H has n vertices, m edges, and there are K graphs in $\mathcal{P}(H)$. In our running time analysis, we will use $r_{\mathcal{P}}(n, m, K)$ to denote the running time of $\textsc{GenRestrHered}\langle\mathcal{P}\rangle(H)$. Observe that $r_{\mathcal{P}}$ is a function of both the input and the output.

Remark 5.1 *Sometimes, we will use $r_{\mathcal{P}}(n', m', K')$, where $n' \geq n$, $m' \geq$*

m, $K' \geq K$, to denote the function $r_{\mathcal{P}}(n, m, K)$, where n', m' and K' are plugged-in instead of n, m and K, respectively. This will be useful when we only have an upper bound on n, m and K. Note that sometimes $r_{\mathcal{P}}(n, m, K)$ may be a function of n and m alone (e.g., for the graph property $\mathcal{P}_{\text{STAR}}$). If this is the case, and K' is greater than the size of the actual output of GenRestrHered$\langle\mathcal{P}\rangle(H)$*, then $r_{\mathcal{P}}(n', m', K')$ may be less than K'.*

Throughout this article, we will assume that GenRestrHered is implemented as a *coroutine*, i.e., as an algorithm that stores its internal state and that can be paused and restarted after each result is returned.[5] Thus, for example, a loop over the results of GenRestrHered (see Line 2 in Figure 2) is not implemented by computing and storing all of the results of GenRestrHered and then iterating over these stored results. Instead, GenRestrHered is run until a single result is returned, is then paused, and the content of the loop is executed. Afterwards, once again GenRestrHered is run until a single result is returned, and so on. Note that implementing GenRestrHered as a coroutine is not simply a programming design choice. It is in fact necessary in order to achieve our complexity results.

5.1 Polynomial Total Time, Incremental Polynomial Time and Polynomial Delay Algorithms

In this section we show that the maximal $\mathcal{P}$-subgraphs problem is solvable in polynomial total time if and only if the input-restricted maximal $\mathcal{P}$-subgraphs problem is solvable in polynomial total time. Similarly, we show that the maximal $\mathcal{P}$-subgraphs problem is solvable in incremental polynomial time if and only if the input-restricted maximal $\mathcal{P}$-subgraphs problem is solvable in incremental polynomial time. For polynomial delay we show a weaker result, namely, the maximal $\mathcal{P}$-subgraphs problem is solvable in polynomial delay if the input-restricted maximal $\mathcal{P}$-subgraphs problem is solvable in polynomial time.

The procedure RecursiveGen$\langle\mathcal{P}\rangle$ (Figure 2) is used to reduce the maximal $\mathcal{P}$-subgraphs problem to the input-restricted version of this problem. RecursiveGen$\langle\mathcal{P}\rangle$ uses a balanced tree $\mathcal{B}$ to store previously printed graphs. The balanced tree $\mathcal{B}$ is a global variable that is changed during the recursive calls of the procedure, and is initially empty.

RecursiveGen$\langle\mathcal{P}\rangle$ uses the procedure GenRestrHered$\langle\mathcal{P}\rangle$ describes above, as well as the procedure ExtendMax$\langle\mathcal{P}\rangle(H, G)$. This simple procedure, which appears in the appendix in Figure A.1, gets graphs H and G such that $H \in \mathcal{P}$

[5] A precise definition of a coroutine and its implementation is discussed in [39,40]. In that paper, coroutines are called *threaded enumerators.*

```
RecursiveGen⟨$\mathcal{P}$⟩($G, G_{curr}, i$)
 1  foreach $u$ in $V(G) - V(G_{curr})$
 2      do foreach $H$ in GenRestrHered⟨$\mathcal{P}$⟩($G[G_{curr}, u]$)
 3          do $H^m \leftarrow$ ExtendMax⟨$\mathcal{P}$⟩($H, G$)
 4             if $H^m \notin \mathcal{B}$
 5                then if $i$ is even
 6                        then Print($H^m$)
 7                     Insert($H^m, \mathcal{B}$)
 8                     RecursiveGen⟨$\mathcal{P}$⟩($G, H^m, i+1$)
 9                     if $i$ is odd
10                        then Print($H^m$)
```

Fig. 2. Procedure that reduces the maximal $\mathcal{P}$-subgraphs problem to the input-restricted version of this problem.

and $H \sqsubseteq G$. It returns a maximal graph H' such that $H \sqsubseteq H'$ and $H' \in \mathcal{P}(G)$. Intuitively, ExtendMax⟨$\mathcal{P}$⟩ adds vertices to H while preserving property $\mathcal{P}$, until a maximal graph is derived. ExtendMax⟨$\mathcal{P}$⟩ runs in time $\mathcal{O}(n\, s_{\mathcal{P}}(n,m))$ if $\mathcal{P}$ is hereditary and in time $\mathcal{O}(n\, s_{\mathcal{P}}(n,m)+m)$ if $\mathcal{P}$ is connected-hereditary. (Recall that $s_{\mathcal{P}}(n,m)$ denotes the amount of time needed to check if $G \in \mathcal{P}$, i.e., the running time of Sat⟨$\mathcal{P}$⟩(G).) A detailed running time analysis appears in the appendix.

We note that RecursiveGen⟨$\mathcal{P}$⟩ receives three parameters, G, G_{curr} and i. Initially any $\mathcal{P}$-subgraph of G can be provided for G_{curr}. Thus, for example, we can use the empty graph (i.e., the graph with no edges or vertices) as G_{curr}. Choosing a different G_{curr} will not affect the output of the algorithm. We also note that any integer value can be used for i. This parameter is used to control the timing of the output, in the spirit of [24]. We will show that for a special case, this allows our algorithm to run in polynomial delay (Theorem 5.5).

The next theorem specifies exactly the output of RecursiveGen.

Theorem 5.2 (Output of RecursiveGen⟨$\mathcal{P}$⟩) *Let $\mathcal{P}$ be a hereditary or connected-hereditary property. Let G be a graph and G_{curr} be a $\mathcal{P}$-subgraph of G. Let i be an integer. Then, after calling* RecursiveGen⟨$\mathcal{P}$⟩(G, G_{curr}, i), *every graph in $\mathcal{P}(G)$ will be printed exactly once.*

Proof. First, observe that by definition of ExtendMax⟨$\mathcal{P}$⟩, only graphs belonging to $\mathcal{P}(G)$ will be printed. Second, it is also easy to see that each graph $\mathcal{P}(G)$ will be printed at most once, because of the test in Line 4. Hence, it remains to show that every graph satisfying the conditions of the theorem will be printed at least one time.

Let H and H' be induced subgraphs of G. We use $H \cap_{\mathcal{P}} H'$ to denote the graph H^* such that

- $H^* = G[V(H) \cap V(H')]$ if $\mathcal{P}$ is hereditary;
- H^* is a maximum-size connected induced subgraph of $G[V(H) \cap V(H')]$ if $\mathcal{P}$ is connected hereditary.

Note that $G[V(H) \cap V(H')]$ may not be connected. However, if $\mathcal{P}$ is connected hereditary, then H^* is a largest-size connected component of $G[V(H) \cap V(H')]$.

Now, consider a graph $H \in \mathcal{P}(G)$. We will show that H is printed during the execution of RecursiveGen$\langle\mathcal{P}\rangle$. Let H' be a graph that is inserted into $\mathcal{B}$ (and hence, is printed) during the execution of RecursiveGen$\langle\mathcal{P}\rangle$ such that $H \cap_{\mathcal{P}} H'$ is of maximal size (i.e., has a maximal number of vertices). If $H = H'$, then we are finished. Suppose, by way of contradiction, that $H \neq H'$.

Since $H \neq H'$, there is a vertex $w \in V(H) - V(H')$ such that $G[H \cap_{\mathcal{P}} H', w]$ satisfies $\mathcal{P}$. Now, consider the point in the execution of the procedure immediately after H' was generated and inserted into $\mathcal{B}$, i.e., the recursive call to RecursiveGen$\langle\mathcal{P}\rangle(G, H', i')$. The vertex w is not in H'. Therefore, w is one of the vertices returned in Line 1. Clearly, a graph $H'' \sqsubseteq G[H', w]$ that subsumes $G[H \cap_{\mathcal{P}} H', w]$ and satisfies $\mathcal{P}$ will be created in Line 3. This graph is either in $\mathcal{B}$, or will be inserted into $\mathcal{B}$ in Line 7. However, this contradicts the maximality of H', since $H \cap_{\mathcal{P}} H''$ contains at least one vertex more than $H \cap_{\mathcal{P}} H'$. □

We now analyze the total running time of RecursiveGen$\langle\mathcal{P}\rangle$.

Theorem 5.3 (Total Running Time) *Let G be a graph with n vertices and m edges. Let G_{curr} be $\mathcal{P}$-subgraph of G and i be an integer. Let K be the number of graphs in $\mathcal{P}(G)$. The running time of* RecursiveGen$\langle\mathcal{P}\rangle(G, G_{curr}, i)$ *is:*

$$\begin{array}{ll} \mathcal{O}(n^2\, K\, r_{\mathcal{P}}(n,m,K)(s_{\mathcal{P}}(n,m)+n)) & \textit{if } \mathcal{P} \textit{ is hereditary} \\ \mathcal{O}(n^2\, K\, r_{\mathcal{P}}(n,m,nK)(s_{\mathcal{P}}(n,m)+n)) & \textit{if } \mathcal{P} \textit{ is connected hereditary} \end{array}$$

Proof. It follows from Parts 2 and 4 of Proposition 3.2 that the cumulative running time of GenRestrHered$\langle\mathcal{P}\rangle$ (when called within RecursiveGen$\langle\mathcal{P}\rangle$), is a function of the form $r_{\mathcal{P}}(n,m,K)$ or $r_{\mathcal{P}}(n,m,nK)$ (depending on whether $\mathcal{P}$ is hereditary or connected hereditary), where K is the total number of graphs returned by RecursiveGen$\langle\mathcal{P}\rangle$.

We can now analyze the running time of RecursiveGen$\langle\mathcal{P}\rangle$. We first show the running time for the case where $\mathcal{P}$ is hereditary. It follows from Theorem 5.2 that the condition of Line 4 is true exactly K times. Therefore, there are, in total, $K+1$ calls to RecursiveGen$\langle\mathcal{P}\rangle$ (the first call, and then K

recursive calls).

Line 1 returns at most n vertices. For each of these vertices, the procedure GenRestrHered in Line 2 is executed, which takes time $r_{\mathcal{P}}(n, m, K)$ if $\mathcal{P}$ is hereditary (by the definition of $r_{\mathcal{P}}$ and by Part 2 of Proposition 3.2). Thus, throughout the entire runtime, Line 2 requires $\mathcal{O}(n\,K\,r_{\mathcal{P}}(n, m, K))$ time.

Each iteration of Line 2 returns at most $r_{\mathcal{P}}(n, m, K)$ graphs. Since Line 2 is executed $\mathcal{O}(n\,K)$ times in total, Line 3 is executed $\mathcal{O}(n\,K\,r_{\mathcal{P}}(n, m, K))$ times. The procedure ExtendMax$\langle\mathcal{P}\rangle$ runs in time $\mathcal{O}(n\,s_{\mathcal{P}}(n, m))$, when $\mathcal{P}$ is hereditary. Hence, throughout the entire runtime, Line 3 requires a total of $\mathcal{O}(n\,K\,r_{\mathcal{P}}(n, m, K)(n\,s_{\mathcal{P}}(n, m)))$ time.

Line 4 is also executed $\mathcal{O}(n\,K\,r_{\mathcal{P}}(n, m, K))$ times in total. Since $\mathcal{B}$ can contain at most 2^n graphs, it is possible to check if H^m is in $\mathcal{B}$ in time $\mathcal{O}(n^2)$ (since the depth of $\mathcal{B}$ is n and n comparisons are needed to compare graphs). Thus, throughout the entire runtime, Line 4 requires $\mathcal{O}(n^3\,K\,r_{\mathcal{P}}(n, m, K))$ time.

Lines 5–10 are only executed if $H^m \notin \mathcal{B}$. This is true exactly K times. Hence, the insertion into $\mathcal{B}$, which takes time $\mathcal{O}(n^2)$, is executed exactly K times in total.

From the above analysis we derive that the total running time is

$$\mathcal{O}(n\,K\,r_{\mathcal{P}}(n, m, K)(n\,s_{\mathcal{P}}(n, m) + n^2)) = \mathcal{O}(n^2\,K\,r_{\mathcal{P}}(n, m, K)(s_{\mathcal{P}}(n, m) + n))\,,$$

as required.

For the case that $\mathcal{P}$ is connected-hereditary, almost exactly the same complexity analysis is applicable. The only changes are that *(1)* the running time of GenRestrHered$\langle\mathcal{P}\rangle$ is a function of the form $r_{\mathcal{P}}(n, m, nK)$ *and (2)* ExtendMax$\langle\mathcal{P}\rangle$ runs in time $\mathcal{O}(n\,s_{\mathcal{P}}(n, m) + m)$. By plugging-in these changes, and recalling that $m \leq n^2$, we derive the required running time. $\square$

Characterization 1 follows immediately from Theorems 5.2 and 5.3.

Characterization 1 *For all hereditary or connected-hereditary properties $\mathcal{P}$, the maximal $\mathcal{P}$-subgraphs problem is solvable in polynomial total time if and only if the input-restricted maximal $\mathcal{P}$-subgraphs problem is solvable in polynomial total time.*

We show a characterization for the complexity class incremental polynomial time. To do so, we show that RecursiveGen$\langle\mathcal{P}\rangle$ runs in incremental polynomial time if GenRestrHered$\langle\mathcal{P}\rangle$ runs in incremental polynomial time.

Theorem 5.4 (Incremental Polynomial Time) *Let $\mathcal{P}$ be a hereditary or* *connected-hereditary property. If* GenRestrHered$\langle\mathcal{P}\rangle$ *runs in incremental polynomial time, then* RecursiveGen$\langle\mathcal{P}\rangle$ *runs in incremental polynomial time.*

Proof. Assume that GenRestrHered$\langle\mathcal{P}\rangle$ runs in incremental polynomial time. Suppose that k graphs have been printed by RecursiveGen$\langle\mathcal{P}\rangle$. We analyze the number of graphs that may be produced and used in a recursive call to RecursiveGen$\langle\mathcal{P}\rangle$ before a new graph is printed. If we can show that this number is polynomial in k and n, then the theorem follows by the same complexity analysis as in Theorem 5.3.

Since k graphs have been printed,

(1) there have been at most $2k+1$ recursive calls to RecursiveGen$\langle\mathcal{P}\rangle$ *and*
(2) there are at most $2k$ graphs in $\mathcal{B}$.

To see why these claims hold, observe that a call to RecursiveGen$\langle\mathcal{P}\rangle$ is made in the very beginning, and then each time that a new graph H, not appearing in $\mathcal{B}$, is found. When the depth of the recursion is even, the graph H is printed before the call to RecursiveGen$\langle\mathcal{P}\rangle$. When the depth of the recursion is odd, H is printed after calling RecursiveGen$\langle\mathcal{P}\rangle$, but before any subsequent call to RecursiveGen$\langle\mathcal{P}\rangle$. Let j be the recursion depth at the point when k graphs have been printed. By the discussion above, it follows that $j \leq 2k$ (since only k graphs have been printed). Item 1 follows immediately. Item 2 follows from Item 1, since graphs are only added to $\mathcal{B}$ before calling RecursiveGen$\langle\mathcal{P}\rangle$.

Since there have been at most $2k+1$ calls to the procedure RecursiveGen$\langle\mathcal{P}\rangle$, there have been at most $n(2k+1)$ calls to GenRestrHered$\langle\mathcal{P}\rangle$. (Observe that GenRestrHered$\langle\mathcal{P}\rangle$ is called at most n times during each execution of RecursiveGen$\langle\mathcal{P}\rangle$.)

If $\mathcal{P}$ is hereditary, then by Part 1 of Proposition 3.2, each graph in the output subsumes at most one graph in the result of each execution of the procedure GenRestrHered$\langle\mathcal{P}\rangle$. Recall that there are at most $2k$ graphs in $\mathcal{B}$. Hence, at most $2k$ graphs can be produced by any of the $n(2k+1)$ executions of the procedure GenRestrHered$\langle\mathcal{P}\rangle$ before a new graph is returned that is not subsumed by any graph in the output. This graph will result in printing a new graph in the output.

Similarly, if $\mathcal{P}$ is connected hereditary, then by Part 3 of Proposition 3.2, each graph in the output subsumes at most n graphs in the result of each execution of the procedure GenRestrHered$\langle\mathcal{P}\rangle$. Hence, at most $n(2k)$ graphs can be produced by each of the $n(2k+1)$ executions of GenRestrHered$\langle\mathcal{P}\rangle$ before a new graph is returned that is not subsumed in any graph in the output. Once again, this graph will result in printing a new graph in the output. □

Characterization 2 follows from Theorems 5.2 and 5.4.

Characterization 2 *For all hereditary or connected-hereditary properties $\mathcal{P}$, the maximal $\mathcal{P}$-subgraphs problem is solvable in incremental polynomial time if and only if the input-restricted maximal $\mathcal{P}$-subgraphs problem is solvable in incremental polynomial time.*

We now show that for the special case in which the input-restricted $\mathcal{P}$-subgraphs problem is solvable in PTIME (e.g., $\mathcal{P}_{\text{CLIQUE}}$, $\mathcal{P}_{\text{STAR}}$), the maximal $\mathcal{P}$-subgraphs problem can be solved in polynomial delay.

Theorem 5.5 (Polynomial Delay) *Let $\mathcal{P}$ be a hereditary or a connected-hereditary property. If* GENRESTRHERED$\langle\mathcal{P}\rangle$ *runs in polynomial time, then* RECURSIVEGEN$\langle\mathcal{P}\rangle$ *runs in polynomial delay.*

Proof. The execution of RECURSIVEGEN$\langle\mathcal{P}\rangle(G, G_{curr}, i)$ can be abstractly described as a traversal of the recursion (i.e., execution) tree of the procedure. Suppose that directly within the execution of RECURSIVEGEN$\langle\mathcal{P}\rangle(G, G_{curr}, i)$, recursive calls are made with graphs $H_1, \dots, H_k$, i.e., for $j \leq k$, recursive calls are made to RECURSIVEGEN$\langle\mathcal{P}\rangle(G, H_j, i+1)$, in that order. Then, the *recursion tree* $\mathcal{R}_{\mathcal{P}}^{G,G_{curr}}$ of RECURSIVEGEN$\langle\mathcal{P}\rangle(G, G_{curr}, i)$ is defined in the following manner:

- the root of $\mathcal{R}_{\mathcal{P}}^{G,G_{curr}}$ is G_{curr};
- the children of $\mathcal{R}_{\mathcal{P}}^{G,G_{curr}}$ are the trees $\mathcal{R}_{\mathcal{P}}^{G,H_1}, \dots, \mathcal{R}_{\mathcal{P}}^{G,H_k}$, in this order.

Observe that RECURSIVEGEN$\langle\mathcal{P}\rangle(G, G_{curr}, i)$ performs a depth-first traversal of $\mathcal{R}_{\mathcal{P}}^{G,G_{curr}}$. During the traversal, nodes at even levels are printed as they are reached, whereas nodes at odd levels are printed as we return from the depth-first traversal of their children.

Assuming that GENRESTRHERED$\langle\mathcal{P}\rangle$ runs in polynomial time, it follows that *(1)* each node has a polynomial number of children *and (2)* the children of each node can be generated in polynomial time. Now, it immediately follows that graphs are printed with polynomial delay, since every other traversal step (either upwards or downwards) will cause a graph to be printed. □

Sufficient Condition 1 follows immediately.

Sufficient Condition 1 *For all hereditary or connected-hereditary properties $\mathcal{P}$, the maximal $\mathcal{P}$-subgraphs problem is solvable in polynomial delay if the input-restricted maximal $\mathcal{P}$-subgraphs problem is solvable in* PTIME.

Although RecursiveGen$\langle\mathcal{P}\rangle$ achieves an efficient running time, it requires memory that is linear in the size of the output. Since the output may be large, this is a clear drawback. In this section we present a polynomial total time PSPACE algorithm that computes $\mathcal{P}(G)$ for *hereditary properties* $\mathcal{P}$.[6] For the special case that the input-restricted maximal $\mathcal{P}$-subgraphs problem is solvable in polynomial time, this algorithm runs in polynomial delay. The basic strategy that we will employ is to formulate the maximal $\mathcal{P}$-subgraphs problem as a tree search problem, by using a *lexicographic traversal tree.* This idea is similar in spirit to other enumeration algorithms that can be described as graph traversals, e.g. [23].

In order to use only linear space, there must be a way to ensure that each graph is printed only once, without storing all the graphs that have already been produced. This is accomplished by defining an (arbitrary) ordering on the vertices of the graph, and then utilizing this ordering to determine whether a graph should be printed. Formally, if G has n vertices, we denote the vertices of G as $v_1, \ldots, v_n$. The indices of the vertices indicate their relative ordering. We use G_i to denote the graph $G[\{v_1, \ldots, v_i\}]$. We use $\mathcal{G}_i$ to denote the set of maximal $\mathcal{P}$-subgraphs of G_i, i.e., $\mathcal{G}_i := \mathcal{P}(G_i)$.

The ordering on the vertices of a graph G defines a lexicographic ordering on the induced subgraphs of G. Formally, let H_1 and H_2 be different induced subgraphs of G. Let v be the first vertex (according to the ordering of $V(G)$) that is in one of the graphs H_1 and H_2, but not in both graphs. We say that H_1 *precedes* H_2, if $v \in V(H_1)$. Otherwise, H_2 precedes H_1.

Let $\mathcal{P}$ be a hereditary property and let G be a graph with n vertices. We define the *lexicographic traversal tree* of G, denoted by $\mathcal{T}_{\mathcal{P}}^G$. The tree $\mathcal{T}_{\mathcal{P}}^G$ is of depth n, and for each $H \in \mathcal{G}_i$, it has a vertex W_H^i at depth i. Intuitively, we use the notation W_H^i to indicate a node of depth i that is associated with a graph H in $\mathcal{G}_i$. Finally, the tree-like structure is determined in the following manner. For $i > 1$, a vertex W_H^i is a child of the vertex $W_{H'}^{i-1}$ if H' is the lexicographically first graph in $\mathcal{G}_{i-1}$ such that $H - v_i \sqsubseteq H'$. (Note that such a graph H' always exists since $\mathcal{P}$ is hereditary, and thus, $H - v_i \in \mathcal{P}$ and $H - v_i$ is subsumed by some graph $H' \in \mathcal{G}_{i-1}$.)

The lexicographic traversal tree of G has the following important property.

[6] This algorithm is similar in spirit to the algorithm of [14]. However, the algorithm of [14] is applicable only for generating maximal independent sets, whereas the algorithm presented here is applicable for general hereditary properties. In addition, our algorithm has a simpler formulation.

Proposition 5.6 *Let G be a graph and $\mathcal{P}$ be a hereditary property. All leaf vertices of $\mathcal{T}_{\mathcal{P}}^{G}$ are of depth $|V(G)|$.*

Proof. Let W_H^i be a vertex in $\mathcal{T}_{\mathcal{P}}^{G}$, where $i < |V(G)|$. We consider two cases and show that in both cases W_H^i must have at least one child.

First, suppose that $G[H, v_{i+1}] \in \mathcal{P}$. It then follows that $G[H, v_{i+1}] \in \mathcal{G}_{i+1}$ and that H is the lexicographically first graph in $\mathcal{G}_i$ such that $G[H, v_{i+1}] - v_{i+1} \sqsubseteq H$. Therefore, $W_{G[H,v_{i+1}]}^{i+1}$ is a child of W_H^i.

Second, suppose that $G[H, v_{i+1}] \notin \mathcal{P}$. It then follows that $H \in \mathcal{G}_{i+1}$ and that H is the lexicographically first graph in $\mathcal{G}_i$ such that $H - v_{i+1} = H \sqsubseteq H$. (Note that the only graph H' of level i for which $H \sqsubseteq H'$ is the graph H itself, due to the maximality of the graphs at level i.) Therefore, W_H^{i+1} is a child of W_H^i.
□

Corollary 5.7 *Let G be a graph and $\mathcal{P}$ be a hereditary property. Then $\mathcal{P}(G)$ is determined exactly by the set of leaves in $\mathcal{T}_{\mathcal{P}}^{G}$.*

Proof. The claim follows directly from Proposition 5.6 and from the fact that (by definition of $\mathcal{T}_{\mathcal{P}}^{G}$) there is a one-to-one correspondence between the vertices W_H^n at depth n and the graphs of $\mathcal{P}(G)$. □

From Corollary 5.7, we conclude that $\mathcal{P}(G)$ may be enumerated by traversing the tree $\mathcal{T}_{\mathcal{P}}^{G}$, if this tree can be generated. Therefore, we now consider the problem of generating $\mathcal{T}_{\mathcal{P}}^{G}$.

It is easy to generate the root of $\mathcal{T}_{\mathcal{P}}^{G}$, since the root is simply $W_{G[v_1]}^1$, if $G[v_1] \in \mathcal{P}$, or the empty graph (with no vertices or edges), otherwise. Now, consider some vertex W_H^i in $\mathcal{T}_{\mathcal{P}}^{G}$, where $i < |V(G)|$. Then, to generate the children of W_H^i, we must find all graphs $H' \in \mathcal{G}_{i+1}$ such that H is the lexicographically first element in $\mathcal{G}_i$ with $H' - v_{i+1} \sqsubseteq H$. Observe that $G[H, v_{i+1}]$ almost satisfies $\mathcal{P}$. Hence, we iterate over the children of W_H^i by calling $\textsc{GenRestrHered}\langle\mathcal{P}\rangle(G[H, v_{i+1}])$, to derive each graph H' in $\mathcal{P}(G[H, v_{i+1}])$, and then:

(1) Discarding H' if it is not in $\mathcal{G}_{i+1}$. This can be verified by simply trying to extend H' with each vertex in G_{i+1} that is not already in H', and checking whether the extended graph is in $\mathcal{P}$. See the procedure $\textsc{IsMax}\langle\mathcal{P}\rangle$ in Figure 3 for details.
(2) Discarding H' if there exists a graph $H'' \in \mathcal{G}_i$ that precedes H, and $H' - v_{i+1} \sqsubseteq H''$. This can be checked by calling $\textsc{LexFirst}\langle\mathcal{P}\rangle(H, H' - v_{i+1}, G, i)$ (Figure 4).

Now, suppose that G has n vertices, m edges and that there are K graphs

```
IsMax⟨$\mathcal{P}$⟩$(H, G)$
    ▷ **Input:** Graph $H \in \mathcal{P}$, Graph $G$ such that $H \sqsubseteq G$
    ▷ **Output:** Returns **true** if $H$ is a maximal $\mathcal{P}$-subgraph of $G$
1   foreach $v$ in $V(G) - V(H)$
2       do if Sat⟨$\mathcal{P}$⟩$(G[H, v])$
3           then return false
4   return true
```

Fig. 3. Algorithm that checks if given subgraph is maximal with respect to a graph and property.

```
LexFirst⟨$\mathcal{P}$⟩$(H, H', G, i)$
    ▷ **Input:** Graphs $H$, $H'$, $G$ such that $H' \sqsubseteq H \sqsubseteq G$, index $i$
    ▷ **Output:** Returns **true** if $H$ is the first graph in $\mathcal{G}_i$ s.t. $H' \sqsubseteq H$
1   $W \leftarrow \emptyset$
2   for $j = 1$ to $i$
3       do if $v_j \in V(H)$
4           then $W \leftarrow W \cup \{v_j\}$
5           else if Sat⟨$\mathcal{P}$⟩$(G[W, H', v_j])$
6               then return false
7   return true
```

Fig. 4. Algorithm that checks if H is the lexicographically first graph in $\mathcal{G}_i$ that contains H'.

in $\mathcal{P}(G)$. Then, calling the algorithm GenRestrHered⟨$\mathcal{P}$⟩ has a cost of $\mathcal{O}(r_{\mathcal{P}}(n, m, K))$, and at most $\mathcal{O}(r_{\mathcal{P}}(n, m, K))$ graphs in its output.[7] Both IsMax⟨$\mathcal{P}$⟩ and LexFirst⟨$\mathcal{P}$⟩ run in time $\mathcal{O}(n\, s_{\mathcal{P}}(n, m))$. Thus, given a node H in $\mathcal{T}^G_{\mathcal{P}}$, the total time that will be needed to generate all of the children of H is

$$\mathcal{O}(n\, s_{\mathcal{P}}(n, m)\, r_{\mathcal{P}}(n, m, K)). \tag{1}$$

To generate $\mathcal{P}(G)$, we simply perform a depth-first search of $\mathcal{T}^G_{\mathcal{P}}$ and print all (subscripts of) the leaf vertices. It is important to note that we take advantage of the fact that GenRestrHered is implemented as a coroutine to perform the depth-first search efficiently in time and space. Thus, during the depth-first

[7] By Part 2 of Proposition 3.2, the number of graphs in the output of the procedure GenRestrHered⟨$\mathcal{P}$⟩ is bounded by K, since $G[H, v_{i+1}] \sqsubseteq G$. However, we prefer to use the bound $r_{\mathcal{P}}(n, m, K)$ to simplify the formulation, and since $r_{\mathcal{P}}(n, m, K)$ may be much smaller than K. This may hold, since K is an upper-bound on the size of the output and $r_{\mathcal{P}}$ may possibly be a function of n and m alone. See Remark 5.1 for details.

search, we do not explicitly create and store all children of an intermediate node. Instead we create a single child for the intermediate node, continute the depth-first search recursively at this child, and only upon return from the recursion do we generate the next child.

The following theorem states the running time required to print $\mathcal{P}(G)$ in this fashion.

Theorem 5.8 (Running time to Print $\mathcal{P}(G)$) *Let $\mathcal{P}$ be a hereditary property and G be a graph with n vertices and m edges. Suppose that there are K graphs in $\mathcal{P}(G)$. Then, all graphs in $\mathcal{P}(G)$ can be printed in time*

$$\mathcal{O}(n^2\, K\, s_{\mathcal{P}}(n,m)\, r_{\mathcal{P}}(n,m,K))) .$$

Moreover, if the input-restricted maximal $\mathcal{P}$-subgraph problem is solvable in PTIME*, then the $\mathcal{P}(G)$ can be printed with polynomial delay.*

Proof. We prove each of the claims by analyzing the running time of a depth-first search of $\mathcal{T}_{\mathcal{P}}^{G}$.

Total Running Time. The running time of the depth-first search is simply the time required to generate $\mathcal{T}_{\mathcal{P}}^{G}$. For each $i < n$, and each graph $H \in \mathcal{G}_i$, we must generate all children of H. According to Part 2 of Proposition 3.2, $|\mathcal{G}_i| \leq |\mathcal{G}_n|$ for all $i \leq n$. Hence, there are at most $(n-1)K$ non-leaf vertices in $\mathcal{T}_{\mathcal{P}}^{G}$. Using Formula (1), we can conclude that $\mathcal{T}_{\mathcal{P}}^{G}$ can be generated in time

$$\mathcal{O}(n^2\, K\, s_{\mathcal{P}}(n,m)\, r_{\mathcal{P}}(n,m,K))$$

as required.

Polynomial Delay. Recall that $\mathcal{T}_{\mathcal{P}}^{G}$ has depth n, and that all leaves are in $\mathcal{P}(G)$. Therefore, a depth-first search of $\mathcal{T}_{\mathcal{P}}^{G}$ will visit at most $2n$ vertices between printing answers. Once again, using Formula (1), we can conclude that the running time between printing answers is bounded by

$$\mathcal{O}\Big(n^2\, s_{\mathcal{P}}(n,m)\, r_{\mathcal{P}}(n,m,K)\Big) .$$

Assuming that $r_{\mathcal{P}}(n,m,K)$ is a polynomial in n and m (and does not depend on K), we derive polynomial delay between printing answers. □

We now show that $\mathcal{P}(G)$ can be generated efficiently in terms of time and space.

Theorem 5.9 (Space Requirements) *Suppose that* GenRestrHered$\langle\mathcal{P}\rangle$ *runs in* PSPACE*. Then, $\mathcal{P}(G)$ can be generated in* PSPACE.

Proof. Let G be a graph with n vertices. The depth of $\mathcal{T}_{\mathcal{P}}^{G}$ is n. Thus, at any point in the depth-first traversal of $\mathcal{T}_{\mathcal{P}}^{G}$, it is only necessary to store the state of GenRestrHered for n vertices (i.e., the state of an iteration over the children of each of those vertices). Hence, the space requirement of the depth-first search is n times that of GenRestrHered$\langle\mathcal{P}\rangle$. By our assumption, each execution of GenRestrHered$\langle\mathcal{P}\rangle$ uses only polynomial space. Thus, in total only polynomial space is required. □

Characterization 3 and Sufficient Condition 2 follow from Theorems 5.8 and 5.9.

Characterization 3 *For hereditary properties $\mathcal{P}$, the maximal $\mathcal{P}$-subgraphs problem is solvable in polynomial-total-time* PSPACE *if and only if the input-restricted maximal $\mathcal{P}$ subgraphs problem is solvable in polynomial total time* PSPACE*.*

Sufficient Condition 2 *For hereditary properties $\mathcal{P}$, the maximal $\mathcal{P}$-subgraphs problem is solvable in polynomial-delay* PSPACE *if the input-restricted maximal $\mathcal{P}$ subgraphs problem is solvable in* PTIME*.*

6 Conclusion

We have presented algorithms that reduce the maximal $\mathcal{P}$-subgraphs problem to the input-restricted version of this problem. These algorithms imply that when attempting to efficiently solve the maximal $\mathcal{P}$-subgraphs problem for a specific $\mathcal{P}$, it is not necessary to define an algorithm that works for the general case. Instead, an algorithm for the restricted case should be defined and an efficient method for solving the general case is automatically derived from our algorithms. This approach seems promising, since the input-restricted version is conceptually easier.

Our main results are three characterizations and two sufficient conditions that state when the maximal $\mathcal{P}$-subgraphs problem is in a complexity class C. The advantage of these results is their versatility, i.e., as immediate corollaries of our characterizations one can determine the complexity of the maximal $\mathcal{P}$-subgraphs problem for various properties $\mathcal{P}$. The results can also be used to efficiently evaluate queries over incomplete information using a variety of semantics [30].

Open problems include characterizing when the maximal $\mathcal{P}$-subgraphs problem is in polynomial delay and finding space efficient algorithms for connected-hereditary properties. Another area of interest is to find algorithms that can return solutions in ranking order, for some given ranking function of interest.

Acknowledgments

The authors thank the anonymous referees for helpful suggestions and for pointing out important related work.

References

[1] J. Lewis, On the complexity of the maximum subgraph problem, in: Proc. 10th Annual ACM Symposium on Theory of Computing, ACM Press, New York (USA), 1978, pp. 265–274.

[2] M. Yannakakis, Node- and edge-deletion NP-complete problems, in: Proc. 10th Annual ACM Symposium on Theory of Computing, ACM Press, New York (USA), 1978, pp. 253–264.

[3] J. Lewis, M. Yannakakis, The node-deletion problem for hereditary properties is NP-complete, Journal of Computer and System Sciences 20 (2) (1980) 219–230.

[4] M. Okun, A. Barak, A new approach for approximating node deletion problems, Information Processing Letters 88 (5) (2003) 231–236.

[5] T. Fujito, A unified approximation algorithm for node-deletion problems, Discrete Applied Mathematics 86 (2–3) (1998) 213–231.

[6] C. Lund, M. Yannakakis, The approximation of maximum subgraph problems, in: The 20th International Colloquium on Automata, Languages and Programming, Lund (Sweden), 1993, pp. 40–51.

[7] D. Hochbaum, Approximating clique and biclique problems, Journal of Algorithms 29 (1) (1998) 174–200.

[8] N. Alon, A. Shapira, B. Sudakov, Additive approximation for edge-deletion problems, in: 46th Annual IEEE Symposium on Foundations of Computer Science, Pittsburgh, (Pennsylvania, USA), 2005, pp. 419–428.

[9] Y. Bartal, A. Fiat, S. Leonardi, Lower bounds for on-line graph problems with application to on-line circuit and optical routing, in: Proc. 28th Annual ACM Symposium on Theory of Computing, Philadelphia (Pennsylvania, USA), 1996, pp. 531–540.

[10] J. Balogh, B. Bollobás, D. Weinreich, The speed of growth for hereditary graph properties, Journal of Combinatorial Theory 79 (2000) 131–156.

[11] M. Golumbic, H. Kaplan, R. Shamir, Graph sandwich problems, Journal of Algorithms 19 (3) (1995) 449–473.

[12] A. Puricella, I. Stewart, A generic greedy algorithm, partially-ordered graphs and NP-completeness, in: The 27th International Workshop of Graph-Theoretic Concepts in Computer Science, Boltenhagen (Germany), 2001, pp. 306–316.

[13] D. Johnson, C. Papadimitriou, M. Yannakakis, On generating all maximal independent sets, Information Processing Letters 27 (3) (1988) 119–123.

[14] S. Tsukiyama, M. Ide, H. Ariyoshi, I. Shirakawa, A new algorithm for generating all maximal independant sets, SIAM Journal on Computing 6 (1977) 505–517.

[15] E. Akkoyunlu, The enumeration of maximal cliques of large graphs, SIAM Journal on Computing 2 (1973) 1–6.

[16] K. Makino, T. Uno, New algorithms for enumerating all maximal cliques, in: 9th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science, 2004, pp. 260–272.

[17] R. Read, R. Tarjan, Bounds on backtrack algorithms for listing cycles, paths, and spanning trees, Networks 5 (1975) 237–252.

[18] S. Kapoor, H. Ramesh, Algorithms for enumerating all spanning trees of undirected and weighted graphs, SIAM Journal on Computing 24 (2) (1995) 247–265.

[19] H. Gabow, E. Myers, Finding all spanning trees of directed and undirected trees, SIAM Journal on Computing 7 (1978) 280–287.

[20] S. Nakano, T. Uno, Constant time generation of trees with specified diameter, in: 30th International Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, 2004, pp. 33–45.

[21] D. Babic, A. Graovac, Enumeration of acyclic walks in a graph, Discrete Applied Mathematics 45 (1993) 117–123.

[22] K. Fukuda, T. Matsui, Y. Matsui, Algorithms for combinatorial enumeration problems, `http://dmawww.epfl.ch/roso.mosaic/kf/enum/comb/combenum` (1995).

[23] D. Avis, K. Fukuda, Reverse search for enumeration, Discrete Applied Mathematics 65 (1–3) (1996) 21–46.

[24] T. Uno, Two general methods to reduce delay and change of enumeration algorithms, Tech. Rep. NII-2003-004E, National Institute of Informatics of Japan (April 2003).
URL `http://research.nii.ac.jp/TechReports/03-004E.pdf`

[25] L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, V. Gurvich, K. Makino, Generating cut conjunctions and bridge avoiding extensions in graphs, in: 16th International Symposium on Algorithms and Computation (ISAAC), Lecture Notes in Computer Science, 2005, pp. 156–165.

[26] A. Rajaraman, J. Ullman, Integrating information by outerjoins and full disjunctions, in: Proc. 15th Symposium on Principles of Database Systems, ACM Press, Montreal (Canada), 1996, pp. 238–248.

[27] Y. Kanza, W. Nutt, Y. Sagiv, Queries with incomplete answers over semistructured data, in: Proc. 18th Symposium on Principles of Database Systems, ACM Press, Philadelphia (Pennsylvania, USA), 1999, pp. 227–236.

[28] Y. Kanza, Y. Sagiv, Computing full disjunctions, in: Proc. 22nd Symposium on Principles of Database Systems, ACM Press, San Diego (California, USA), 2003, pp. 78–89.

[29] S. Cohen, Y. Kanza, Y. Sagiv, Generating relations from XML documents, in: Proc 9th International Conference on Database Theory, Springer-Verlag, Siena (Italy), 2003, pp. 285–299.

[30] S. Cohen, Y. Sagiv, An abstract framework for generating all maximal answers to queries, in: Proc 10th International Conference on Database Theory, Edinburgh (UK), 2005, pp. 129–143.

[31] C. Galindo-Legaria, Outerjoins as disjunctions, in: Proc. 1994 ACM SIGMOD International Conference on Management of Data, ACM Press, Minneapolis (Minnesota, USA), 1994, pp. 348–358.

[32] S. Cohen, I. Fadida, Y. Kanza, B. Kimelfeld, Y. Sagiv, Full disjunctions: Polynomial-delay iterators in action, in: Proc. 32nd International Conference on Very Large Data Bases, Morgan Kaufmann Publishers, Seoul (Korea), 2006, pp. 739–750.

[33] B. Kimelfeld, Y. Sagiv, Maximally joining probabilistic data, in: Proc. 26th Symposium on Principles of Database Systems, Beijing (China), 2007, pp. 303–312.

[34] N. Dalvi, D. Suciu, Efficient query evaluation on probabilistic databases, in: Proc. 30th International Conference on Very Large Data Bases, Toronto (Canada), 2004, pp. 864–875.

[35] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[36] L. Valiant, The complexity of enumeration and reliability problems, SIAM Journal on Computing 8 (3) (1979) 410–421.

[37] M. Yannakakis, Algorithms for acyclic database schemas, in: W. Chu, G. Gardarin, S. Ohsuga, Y. Kambayashi (Eds.), Proc. 7th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers, Cannes (France), 1981, pp. 82–94.

[38] L. Valiant, The complexity of computing the permanent, Theoretical Computer Science 8 (1979) 189–201.

[39] B. Kimelfeld, Y. Sagiv, Efficiently enumerating results of keyword search, in: 10th International Workshop on Database Programming Languages, Trondheim (Norway), 2005, pp. 58–73.

[40] B. Kimelfeld, Y. Sagiv, Efficiently enumerating results of keyword search over data graphs, Information Systems 33 (4-5) (2008) 335–359.

A Auxiliary Procedures

A.1 ExtendMax

ExtendMax$\langle\mathcal{P}\rangle(H, G)$: This simple procedure, which appears in the Appendix in Figure A.1, gets graphs H and G such that $H \in \mathcal{P}$ and $H \sqsubseteq G$. It returns a maximal graph H' such that $H \sqsubseteq H'$ and $H' \in \mathcal{P}(G)$. Intuitively, ExtendMax$\langle\mathcal{P}\rangle$ adds vertices to H while preserving property $\mathcal{P}$, until a maximal graph is derived. We sometimes use this procedure to generate a single graph in $\mathcal{P}(G)$ by calling ExtendMax$\langle\mathcal{P}\rangle(O_0, G)$, where O_0 is the empty graph, i.e., a graph with no vertices or edges.

The execution of the procedure differs depending on whether $\mathcal{P}$ is hereditary or connected hereditary. For hereditary properties, the algorithm simply tries to add vertices to H, and checks if the resulting graph satisfies $\mathcal{P}$. It is easy to see that ExtendMax$\langle\mathcal{P}\rangle$ runs in time $\mathcal{O}(n\, s_{\mathcal{P}}(n, m))$ if $\mathcal{P}$ is hereditary. (Recall that $s_{\mathcal{P}}(n, m)$ denotes the amount of time needed to check if $G \in \mathcal{P}$, i.e., the running time of Sat$\langle\mathcal{P}\rangle(G)$.)

For connected-hereditary properties, ExtendMax$\langle\mathcal{P}\rangle$ continuously looks at neighboring vertices and attempts to use them to extend the current graph. It is sufficient to consider each neighbor one time. Formally, in order to produce H', ExtendMax$\langle\mathcal{P}\rangle$ uses a queue $\mathcal{Q}$. In the beginning, $\mathcal{Q}$ is initialized with all the vertices of H. All the vertices of H are marked as *visited*. Then, each vertex v is removed from $\mathcal{Q}$, and each neighbor w of v is considered. If w has not yet been visited, then we check if the current graph can be extended with w. If this is possible, we add w to the current graph. In addition, we add w to $\mathcal{Q}$ so that we will eventually check whether the neighbors of w can be used to extend the current graph.

ExtendMax$\langle\mathcal{P}\rangle$ runs in time $\mathcal{O}(n\, s_{\mathcal{P}}(n, m)+m)$ if $\mathcal{P}$ is connected-hereditary, since *(1)* we examine each vertex to check if it has been visited at most as many times as it has neighbors (which contributes the factor m) *and (2)* we check whether an extended graph satisfies $\mathcal{P}$ at most as many times as there are vertices in G (which contributes the factor $n\, s_{\mathcal{P}}(n, m)$).

```
EXTENDMAX$\langle\mathcal{P}\rangle(H, G)$
    $\triangleright$ **Input:** Graph $H \in \mathcal{P}$, Graph $G$ such that $H \sqsubseteq G$
    $\triangleright$ **Output:** Returns a graph $H'$ such that
    $\triangleright$                 *(1)* $H \sqsubseteq H' \sqsubseteq G$ *and*
    $\triangleright$                 *(2)* $H'$ is a maximal $\mathcal{P}$-subgraph of $G$
    **if** $\mathcal{P}$ is hereditary
      **then** $W \leftarrow V(H)$
           **foreach** $v$ **in** $V(G) - V(H)$
               **do if** SAT$\langle\mathcal{P}\rangle(G[W, v])$
                    **then** $W \leftarrow W \cup v$
      **else** $W \leftarrow V(H)$
           **foreach** $v$ **in** $V(G)$
               **do if** $v \in V(H)$
                    **then** visited($v$)$\leftarrow$ **true**
                    **else** visited($v$)$\leftarrow$ **false**
           ENQUEUE($V(H), \mathcal{Q}$)
           **while** NOTEMPTY($\mathcal{Q}$)
               **do** $v \leftarrow$ DEQUEUE($\mathcal{Q}$)
                  **foreach** $w$ **in** NEIGHBOR$\langle\mathcal{P}\rangle(G[v], G)$
                       **do if** visited($w$) = **false**
                            **then** visited($w$)$\leftarrow$ **true**
                                 **if** SAT$\langle\mathcal{P}\rangle(G[W, w])$
                                   **then** $W \leftarrow W \cup \{w\}$
                                        ENQUEUE($w, \mathcal{Q}$)
    **return** $G[W]$
```

Fig. A.1. An algorithm that finds a maximal graph containing a given subgraph with a given property.