Automated generation of conjectures on forbidden subgraph characterization

Christian Desrosiers
École de Technologie Supérieure
christian.desrosiers@etsmtl.ca

Philippe Galinier
CRT and École Polytechnique de Montréal
philippe.galinier@polymtl.ca

Pierre Hansen
GERAD and HEC Montréal
pierre.hansen@gerad.ca

Alain Hertz
GERAD and École Polytechnique de Montréal
alain.hertz@gerad.ca

Abstract

Given a class of graphs $\mathcal{F}$, a forbidden subgraph characterization (FSC) is a set of graphs $\mathcal{H}$ such that a graph G belongs to $\mathcal{F}$ if and only if no graph of $\mathcal{H}$ is isomorphic to an induced subgraph of G. FSCs play a key role in graph theory, and are at the center of many important results obtained in that field. In this paper, we present novel methods that automate the generation of conjectures on FSCs. Since most classes of graphs do not have such characterization, we also describe methods to find less restrictive results in the form of necessary or sufficient conditions to characterize a class of graphs with forbidden subgraphs. Furthermore, while these methods require to explore a possibly infinite search space, we present an enumerative technique that guarantees the discovery of characterizations involving forbidden subgraphs with a limited number of vertices. Another technique, which enables the discovery of characterizations with much larger subgraphs through the use of heuristic search, is also described. In our experiments, we use these methods to find new theorems on the characterization of well-known graph classes.

Keywords: Graph theory, forbidden subgraph characterization, automated conjecture generation.

1 Introduction

Traditionally, the process of discovering new knowledge in graph theory was carried out by mathematicians, with little assistance from computers. In recent

years, mathematicians in that field have turned to computers to find some very important results. A famous illustration of this is the proof of the four color conjecture, which was done in large part using computers [2, 22]. Since then, computers have played an increasing role in the discovery of new knowledge in graph theory, and many tools have been proposed for this task. One of the first computer programs for this purpose is Graffiti, developed by Fajtlowicz [8], which has generated over a thousand conjectures as algebraic equations involving graph invariants. Another more recent and equally prolific tool to generate conjectures involving graph invariants is AutoGraphiX (AGX), proposed by Caporossi and Hansen [5]. This last program, which applies the Variable Neighborhood Search metaheuristic [20] to find extremal graphs, can also be used to find graphs satisfying various constraints, to find structural conjectures, to refute conjectures and to suggest proofs.

Although automating the generation of conjectures has been the aim of many works, almost all of these focused on generating conjectures in the form of relations on graph invariants. Yet, as recently suggested by Hansen et al. in [16], there are many interesting results in graph theory that take a different form. One of them, known as *forbidden subgraph characterization* (FSC), describes a class of graphs in terms of the subgraphs that these graphs are not allowed to have. A well known FSC, due to Chudnovsky et al. [6], characterizes perfect graphs as the graphs which do not have as induced subgraph any odd cycle containing five or more vertices, or its complement. Another important FSC, due to Beineke [3], characterizes line graphs using nine forbidden graphs, shown in Figure 1.

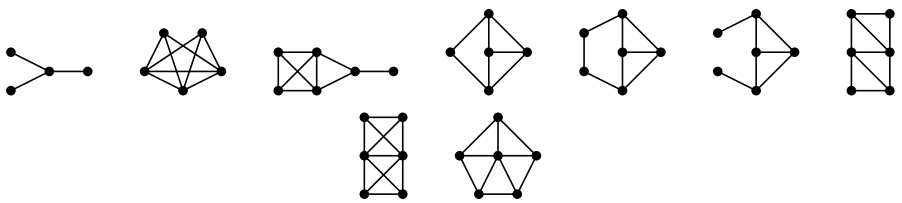


Figure 1: A forbidden subgraph characterization of line graphs.

In this paper, we present methods to automatically generate conjectures on FSCs. To our knowledge, these methods are the first to be proposed for this specific problem. The rest of the paper is structured as follows. We first introduce some preliminary concepts that will help to understand the rest of the paper. We then describe our methods, by considering three problems: finding sufficient conditions of a characterization involving forbidden subgraphs, finding necessary conditions also using forbidden subgraphs, and finding actual FSCs. We then show how these methods can be used in practice to generate conjectures, and illustrate this by reproducing some known results, as well as generating new ones. Finally, we end this paper with a short summary of our work.

2 Preliminary concepts and definitions

Since this work focuses specifically on characterizations involving forbidden *induced* subgraphs, to lighten the presentation, we will from now on refer to

induced subgraphs simply as subgraphs.

Let $\mathcal{G}$ be the set containing all finite graphs. A class of graphs $\mathcal{F} \subseteq \mathcal{G}$ is a possibly infinite set of graphs that share a common property. This common property could be defined using a predicate function $pred : \mathcal{G} \to \{true, false\}$, such that $\forall G \in \mathcal{G},\ pred(G) = true \iff G \in \mathcal{F}$. For instance, the class of perfect graphs could be expressed using the following predicate:

$$pred(G)\ :\ \text{if}, \forall \text{ subgraphs } H \text{ of } G,\ \chi(H) = \omega(H) \text{ then } true, \text{ else } false, \quad (1)$$

where $\chi(H)$ is the smallest number of colors needed to color the vertices of H so that no two adjacent vertices share the same color, i.e. the *chromatic number* of H, and $\omega(H)$ is the maximum order of a clique in H, i.e. its clique number.

Let $\mathcal{H}$ be a set of graphs, we say that a graph G is $\mathcal{H}$-free if there is no graph of $\mathcal{H}$ isomorphic to one of its subgraphs, and denote by $\mathcal{G}_{\mathcal{H}}$ the set of all such graphs. Using this terminology, an FSC of $\mathcal{F}$ is a set of graphs $\mathcal{H}$ such that $\mathcal{G}_{\mathcal{H}} = \mathcal{F}$. As we will see, not every class of graphs has an FSC. For classes that do not have an FSC, we are often interested in finding some weaker rules allowing to partially characterize the graphs of these classes. These rules come in two forms: *sufficient conditions* (SFSC) and *necessary* conditions (NFSC). Let $\mathcal{F}$ be the class of graphs to characterize and $\mathcal{H}$ be a set of forbidden subgraphs. Sufficient conditions can be expressed as follows: if a graph G is $\mathcal{H}$-free, then it is in $\mathcal{F}$. Thus, a sufficient condition can be stated as $\mathcal{G}_{\mathcal{H}} \subseteq \mathcal{F}$. However, sufficient conditions need not fully describe $\mathcal{F}$. Indeed, if G is not $\mathcal{H}$-free, we cannot use this type of condition to determine if G is in $\mathcal{F}$ or not. On the other hand, necessary conditions can be expressed as follows: if a graph G is in $\mathcal{F}$ then it is $\mathcal{H}$-free or, equivalently, $\mathcal{G}_{\mathcal{H}} \supseteq \mathcal{F}$. Again, necessary conditions may offer only a partial description of $\mathcal{F}$: if G is not in $\mathcal{F}$ then it can either be $\mathcal{H}$-free or not.

Let G be a graph, and $W = \{v_1, v_2, \ldots, v_q\}$ be a subset of $V(G)$. We denote by $G[W]$ or, when the context is clear, by $\langle v_1, v_2, \ldots, v_q \rangle$ the subgraph of G induced by W. Furthermore, let H be another graph, we write $G \simeq H$ when G is isomorphic to H, and $H \subseteq G$ when H is isomorphic to a subgraph of G. The following elementary properties will be used later on to prove more complex results.

Property 1. *Let G_1, G_2, G_3 be three graphs, and $\mathcal{H}_1$, $\mathcal{H}_2$ be two sets of forbidden subgraphs.*

(a). If $G_1 \subseteq G_2$ and $G_2 \subseteq G_3$ then $G_1 \subseteq G_3$.

(b). If $G_1 \nsubseteq G_2$ and $G_3 \subseteq G_2$ then $G_1 \nsubseteq G_3$.

(c). If $G_1 \nsubseteq G_2$ and $G_1 \subseteq G_3$ then $G_3 \nsubseteq G_2$.

(d). If $G_1 \subseteq G_2$ then $\mathcal{G}_{\{G_1\}} \subseteq \mathcal{G}_{\{G_2\}}$.

(e). $\mathcal{G}_{\mathcal{H}_1 \cup \mathcal{H}_2} = \mathcal{G}_{\mathcal{H}_1} \cap \mathcal{G}_{\mathcal{H}_2}$.

(f). If $\mathcal{H}_1 \subseteq \mathcal{H}_2$ then $\mathcal{G}_{\mathcal{H}_1} \supseteq \mathcal{G}_{\mathcal{H}_2}$.

(g). If $G_1 \subseteq G_2$ then $\mathcal{G}_{\{G_1\}} = \mathcal{G}_{\{G_1, G_2\}}$.

3 Sufficient conditions

Formally, we write an SFSC as G is $\mathcal{H}$-free $\Rightarrow$ $G \in \mathcal{F}$. This expression is logically equivalent to $G \in \overline{\mathcal{F}} \Rightarrow \exists H \in \mathcal{H}$ s.t. $H \subseteq G$, where $\overline{\mathcal{F}} = \mathcal{G} \setminus \mathcal{F}$ is the complement of the graph class $\mathcal{F}$. The task of finding an SFSC can thus be defined as follows: find a set of graphs $\mathcal{H}$ such that

$$\forall G \in \overline{\mathcal{F}},\ \exists H \in \mathcal{H} \text{ s.t. } H \subseteq G.$$

While a graph class $\mathcal{F}$ can have many SFSCs, these may not be equally useful. For instance, $\mathcal{H} = \overline{\mathcal{F}}$ is an SFSC of $\mathcal{F}$, but is as complex as the class $\mathcal{F}$ itself. Moreover, let H be the graph composed of a single vertex, $\mathcal{H} = \{H\}$ is an SFSC of $\mathcal{F}$ since H is a subgraph of all graphs of $\overline{\mathcal{F}}$. However, $\mathcal{H}$ offers no real information on $\mathcal{F}$, since each graph of $\mathcal{F}$ also has H as subgraph, i.e. $\mathcal{F} \cap \mathcal{G}_{\mathcal{H}} = \emptyset$. To find useful SFSCs, we need to introduce two partial orders that measure the *tightness* and *simplicity*. The tightness of an SFSC evaluates how well it describes $\mathcal{F}$. Let $\mathcal{H}$, $\mathcal{H}'$ be two SFSCs of $\mathcal{F}$, we say that $\mathcal{H}$ is *tighter* than $\mathcal{H}'$ if $\mathcal{G}_{\mathcal{H}'} \subset \mathcal{G}_{\mathcal{H}}$. Thus, $\mathcal{H}$ is maximally tight if $\mathcal{G}_{\mathcal{H}} = \mathcal{F}$. Furthermore, we say that $\mathcal{H}$ is *simpler* than $\mathcal{H}'$ if $\mathcal{H} \neq \mathcal{H}'$ and, for each $H \in \mathcal{H}$, there is a graph $H' \in \mathcal{H}'$ such that $H \subseteq H'$. The concepts of tightness and simplicity are related: a simpler SFSC can generally be obtained at the cost of reducing its tightness.

From the above definitions, we can impose an additional constraint on the selection of an SFSC $\mathcal{H}$. Suppose $\mathcal{H}$ contains two graphs H and H' such that $H' \subset H$. From Property 1g, we know that $\mathcal{H}' = \mathcal{H} \setminus \{H\}$ is a simpler SFSC of $\mathcal{F}$ that is as tight as $\mathcal{H}$. Following this idea, we can limit our search to the sets of graphs $\mathcal{H}$ with the property that

$$\forall H \in \mathcal{H},\ \nexists H' \in \mathcal{H} \text{ s.t. } H' \subset H.$$

In the rest of the paper, we will call *minimal* any set $\mathcal{H}$ that satisfies this property, and denote $\min(\mathcal{H}) \subseteq \mathcal{H}$ the *largest* (w.r.t. cardinality) minimal subset of $\mathcal{H}$. For any $\mathcal{H}$, $\min(\mathcal{H})$ is unique since it can be obtained by removing from $\mathcal{H}$ all H such that $\exists H' \in \mathcal{H}, H' \subset H$, and because such operation can only be done in a single way.

Proposition 1. *Let $\mathcal{F}$ be a class of graphs. Then $\mathcal{H} = \min(\overline{\mathcal{F}})$ is an SFSC of $\mathcal{F}$ and $\mathcal{H}$ is tighter than any other SFSC of $\mathcal{F}$.*

Proof. Let $\mathcal{H} = \min(\overline{\mathcal{F}})$. We first show that $\mathcal{H}$ is an SFSC of $\mathcal{F}$. Suppose it is not, then there is a graph $G \in \overline{\mathcal{F}} \setminus \mathcal{H}$ such that $\nexists H \in \mathcal{H}$, $H \subset G$. However, G would then be, by definition, in $\mathcal{H}$, which is a contradiction. Let $\mathcal{H}'$ be any minimal SFSC of $\mathcal{F}$ such that there is no SFSC tighter than $\mathcal{H}'$. It remains to show that $\mathcal{H}' = \mathcal{H}$. For each $H' \in \mathcal{H}'$, if $\nexists G \in \overline{\mathcal{F}}$ such that $H' \subseteq G$ then, clearly, $\mathcal{H}' \setminus \{H'\}$ is a tighter SFSC of $\mathcal{F}$, a contradiction. Otherwise, if $H' \in \mathcal{F}$, let $\mathcal{B}$ denote the graphs of $\overline{\mathcal{F}}$ which have H' as subgraph. By definition, $\mathcal{H}'' = \mathcal{H}' \setminus \{H'\} \cup \mathcal{B}$ is an SFSC of $\mathcal{F}$. Furthermore, from Properties 1d and 1e, we have that $\mathcal{G}_{\{H'\}} \subseteq \mathcal{G}_{\mathcal{B}}$ and that $\mathcal{G}_{\mathcal{H}'} \subseteq \mathcal{G}_{\mathcal{H}''}$. Finally, since $H' \notin \mathcal{G}_{\mathcal{H}'}$ and $H' \in \mathcal{G}_{\mathcal{H}''}$, we conclude that $\mathcal{G}_{\mathcal{H}'} \subset \mathcal{G}_{\mathcal{H}''}$ and therefore that $\mathcal{H}''$ is tighter than $\mathcal{H}'$, a contradiction. The remaining case is $H' \in \overline{\mathcal{F}}$. Suppose that $H' \notin \mathcal{H}$, then by definition there is a graph $H \in \mathcal{H}$ such that $H \subset H'$. Furthermore, since $\mathcal{H}'$ is an SFSC of $\mathcal{F}$, there must be a graph $H'' \in \mathcal{H}'$ such that $H'' \subseteq H \subset H'$,

which contradicts the minimality of $\mathcal{H}'$. Therefore, we have that $H' \in \mathcal{H}$, and since $\mathcal{H}$ cannot be tighter than $\mathcal{H}'$, we necessarily have $\mathcal{H}' = \mathcal{H}$. $\square$

3.1 Single graph SFSC

Before we tackle the general task of finding an SFSC having an arbitrary number of graphs, we first consider the simpler case of finding a single subgraph SFSC of a graph class $\mathcal{F}$. In this case, the problem can be formulated as finding a set $\mathcal{H} = \{H\}$ such that $\forall G \in \overline{\mathcal{F}},\ H \subseteq G$. Thus, H is a common subgraph of the graphs in $\overline{\mathcal{F}}$. Furthermore, let H, H' be two graphs such that $H \subset H'$. From Property 1d, we have $\mathcal{G}_{\{H\}} \subset \mathcal{G}_{\{H'\}}$ and therefore that $\{H'\}$ is tighter than $\{H\}$. Hence, if we want to maximize the tightness, we should find a common subgraph H which is maximal w.r.t. inclusion. This principle serves as the main idea of our first algorithm, detailed in Figure 2. Let $\mathcal{L}$ be a finite set of graphs and denote by $\text{maxCS}(\mathcal{L})$ the set of common subgraphs of $\mathcal{L}$ that are maximal under inclusion, the single graph SFSC algorithm searches for a graph H_k which is contained in all graphs of $\overline{\mathcal{F}}$, in the following way. Starting with a graph H_0 chosen in $\overline{\mathcal{F}}$ and a set of representative graphs of $\overline{\mathcal{F}}$ containing only H_0, i.e., $\mathcal{L}_0 = \{H_0\}$, the algorithm searches at every iteration k for a graph $G_k \in \overline{\mathcal{F}}$ which does not contain H_k. If such a graph exists, the algorithm then adds G_k to the representatives of $\overline{\mathcal{F}}$, i.e. $\mathcal{L}_{k+1} := \mathcal{L}_k \cup \{G_k\}$, and finds a new graph H_{k+1}, which is a maximal common subgraph of $\mathcal{L}_{k+1}$. Otherwise, if no such graph exists, we know that H_k is a subgraph of all graphs of $\overline{\mathcal{F}}$, and thus is an SFSC of $\mathcal{F}$.

Single graph SFSC algorithm

Input: A graph class $\mathcal{F}$.
Output: A single graph SFSC $\mathcal{H}$ of $\mathcal{F}$.

Choose any H_0 in $\overline{\mathcal{F}}$;
Let $\mathcal{L}_0 := \{H_0\}$, and $k := 0$;

while $\exists G_k \in \overline{\mathcal{F}} \cap \mathcal{G}_{\{H_k\}}$ **do**
  Let $\mathcal{L}_{k+1} := \mathcal{L}_k \cup \{G_k\}$;
  Choose H_{k+1} in $\text{maxCS}(\mathcal{L}_{k+1})$;
  Let $k := k + 1$;
return $\mathcal{H} = \{H_k\}$;

Figure 2: Algorithm to find an SFSC containing a single forbidden subgraph.

Proposition 2. *Let $\mathcal{F}$ be a graph class such that $\overline{\mathcal{F}}$ is finite. The single graph SFSC algorithm, shown in Figure 2, produces, in a finite number of steps, a common subgraph of $\overline{\mathcal{F}}$ which is maximal w.r.t. inclusion.*

Proof. We first prove that the algorithm terminates in a finite number of steps. Consider any two graphs H_i, H_j such that $i < j$. Since H_j is a common subgraph of $\mathcal{L}_j$, and $G_i \in \mathcal{L}_j$, we have $H_j \subseteq G_i$ (*1*). Moreover, since $G_i \in \overline{\mathcal{F}} \cap \mathcal{G}_{\{H_i\}}$, we know that $H_i \not\subseteq G_i$ (*2*). Combining (*1*) and (*2*), we get $H_i \neq H_j$. Furthermore, since H_i and H_j are maximum common subgraphs of $\mathcal{L}_i$ and $\mathcal{L}_j$, respectively, and since $\mathcal{L}_i \subset \mathcal{L}_j$, we have that $|V(H_j)| \leq |V(H_i)|$. Finally, since there is a finite number of graphs of $|V(H_i)|$ or less vertices, the algorithm will terminate in a finite number of steps.

Next, we prove that the graph H_k returned by the algorithm is a common subgraph of $\overline{\mathcal{F}}$. Suppose it is not. Then, there exists a graph $G_k \in \overline{\mathcal{F}}$ that does not have H_k as subgraph. However, this is a contradiction since the algorithm terminates only when no such graph exists. Finally, we show that H_k is maximal w.r.t. inclusion. Suppose this is not the case. Then, there exists a common subgraph H' of $\overline{\mathcal{F}}$, such that $H_k \subset H'$. Since H' is a common subgraph of $\overline{\mathcal{F}}$, it is also a common subgraph of $\mathcal{L}_k \subseteq \overline{\mathcal{F}}$. However, this contradicts the fact that H_k is a maximum common subgraph of $\mathcal{L}_k$. □

One should note that determining whether there exists a graph $G_k \in \overline{\mathcal{F}}$ that does not contain a forbidden subgraph H_k, i.e., testing if $\overline{\mathcal{F}} \cap \mathcal{G}_{\{H_k\}} = \emptyset$, may be impractical if $\overline{\mathcal{F}}$ is infinite. In Section 6, we present a strategy that can generate conjectures by restricting the space in which we search for G_k.

3.2 Multiple graph SFSC

From Proposition 1, we know that if a graph H, obtained by the single graph SFSC algorithm, is not in $\overline{\mathcal{F}}$, then we can find a tighter SFSC containing more than one forbidden subgraph. Let M be the maximum number of graphs we want to have in the SFSC. If $|\min(\overline{\mathcal{F}})| \leq M$, the algorithm should return $\min(\overline{\mathcal{F}})$ since this SFSC has maximum tightness. Otherwise, we search for an SFSC of $\mathcal{F}$ in the following way. We first find a set $\mathcal{M}$ containing $M-1$ of the smallest graphs of $\min(\overline{\mathcal{F}})$. Since $\mathcal{M} = \min(\mathcal{M}) \subset \min(\overline{\mathcal{F}})$, by definition, $\mathcal{M}$ is an SFSC of $\mathcal{G}_{\mathcal{M}} \supset \mathcal{F}$ that has maximum tightness. We then find a common subgraph H of $\overline{\mathcal{F}} \cap \mathcal{G}_{\mathcal{M}}$. Finally, since a graph of $\mathcal{M}$ may contain H, we return the SFSC $\mathcal{H} = \min(\mathcal{M} \cup \{H\})$. Denote by $\text{findMin}(\mathcal{C}, L)$ a procedure that returns the set $\min(\mathcal{C})$, if $|\min(\mathcal{C})| \leq L$, or otherwise returns L of the smallest graphs of $\min(\mathcal{C})$. The multiple graph SFSC algorithm, shown in Figure 3, finds an SFSC for a graph class $\mathcal{F}$, containing at most M graphs.

Proposition 3. *Let $\mathcal{F}$ be a graph class such that $\overline{\mathcal{F}}$ is finite. The multiple graph SFSC algorithm of Figure 3 produces, in a finite number of steps, an SFSC of $\mathcal{F}$ containing at most M graphs.*

Proof. We first show that the algorithm terminates in a finite number of steps. Clearly, the procedure that returns $\text{findMin}(\overline{\mathcal{F}}, M-1)$ will terminate in at most $M-1$ iterations. Furthermore, from Proposition 2, we know that finding the last graph can be done in a finite number of steps. Thus, the algorithm is finite. We next prove that the set $\mathcal{H} = \min(\mathcal{M} \cup \{H_k\})$ returned by the algorithm is an SFSC of at most M graphs. Consider once again the procedure $\text{findMin}(\overline{\mathcal{F}}, M-1)$. Since we have $|\mathcal{M}_l| = l$, at every iteration l, this procedure will return a set $\mathcal{M}$ containing at most $M-1$ graphs. Furthermore, suppose that $\mathcal{M} \cup \{H_k\}$ is not an SFSC. Then, there exists a graph $G \in \overline{\mathcal{F}}$ that does not have H_k nor any graph of $\mathcal{M}$ as subgraph. However, this is a contradiction since the algorithm terminates only when no such graph exists. Thus, $\mathcal{M} \cup \{H_k\}$ is an SFSC of $|\mathcal{M}| + 1 \leq M$ graphs. Finally, since $\min(\mathcal{M} \cup \{H_k\}) \subseteq \mathcal{M} \cup \{H_k\}$, we have, by definition, that $\mathcal{H}$ is an SFSC of at most M graphs. □

Proposition 4. *Let H be a forbidden subgraph obtained by the single graph SFSC algorithm, and $\mathcal{H} = \mathcal{M} \cup \{H_k\}$ be the multiple graph SFSC returned by the above algorithm. If $H \subset H_k$ then $\mathcal{H}$ is a tighter SFSC than $\{H\}$.*

Multiple graph SFSC algorithm

Input: A graph class $\mathcal{F}$ and an integer $M > 0$.
Output: An SFSC $\mathcal{H}$ of $\mathcal{F}$ containing at most M graphs.

Let $\mathcal{M} := \text{findMin}(\overline{\mathcal{F}}, M-1)$ and choose any H_0 in $\overline{\mathcal{F}} \cap \mathcal{G}_{\mathcal{M}}$;
Let $\mathcal{L}_0 := \{H_0\}$, and $k := 0$;

while $\exists G_k \in \overline{\mathcal{F}} \cap \mathcal{G}_{\mathcal{M} \cup \{H_k\}} = \overline{\mathcal{F}} \cap (\mathcal{G}_{\mathcal{M}} \cap \mathcal{G}_{\{H_k\}})$ **do**
  Let $\mathcal{L}_{k+1} := \mathcal{L}_k \cup \{G_k\}$;
  Choose H_{k+1} in $\text{maxCS}(\mathcal{L}_{k+1})$;
  Let $k := k+1$;
return $\mathcal{H} = \min(\mathcal{M} \cup \{H_k\})$;

Procedure `findMin(`$\mathcal{C}$`,`L`)`

Input: A graph class $\mathcal{C}$ and an integer $L > 0$.
Output: At most L of the smallest graphs of $\min(\mathcal{C})$.

Let $\mathcal{M}_0 := \emptyset$ and $l := 0$;

while $\mathcal{C} \cap \mathcal{G}_{\mathcal{M}_l} \neq \emptyset$ *and* $l < L$ **do**
  Choose $H_l \in \mathcal{C} \cap \mathcal{G}_{\mathcal{M}_l}$ with minimum order ;
  Let $\mathcal{M}_{l+1} := \mathcal{M}_l \cup \{H_l\}$;
  Let $l := l+1$;
return $\mathcal{M} = \mathcal{M}_l$;

Figure 3: Algorithm to find an SFSC containing at most M forbidden subgraphs.

Proof. Since $\{H\}$ is an SFSC and $\mathcal{M} \subseteq \overline{\mathcal{F}}$, we know that $H \subseteq H'$, for all $H' \in \mathcal{M}$, and, from Properties 1d and 1e, that $\mathcal{G}_{\{H\}} \subseteq \mathcal{G}_{\mathcal{M}}$. Furthermore, since $H \subset H_k$, we have $\mathcal{G}_{\{H\}} \subset \mathcal{G}_{\{H_k\}}$, and thus that $\mathcal{G}_{\{H\}} \subset \mathcal{G}_{\mathcal{M}} \cap \mathcal{G}_{\{H_k\}} = \mathcal{G}_{\mathcal{H}}$. Therefore $\mathcal{H}$ is a tighter SFSC than $\{H\}$. □

4 Necessary conditions

Let $\mathcal{F}$ be a graph class, an NFSC is a set of graphs $\mathcal{H}$ such that all graphs in $\mathcal{F}$ are $\mathcal{H}$-free. Finding an NFSC thus amounts to finding $\mathcal{H}$ such that

$$\forall G \in \mathcal{F}, \ \nexists H \in \mathcal{H} \text{ s.t. } H \subseteq G.$$

This implies that $\mathcal{H} \subseteq \overline{\mathcal{F}}$. As we did for SFSCs, we can use the criteria of tightness and simplicity to guide our search of an NFSC. However, since an NFSC $\mathcal{H}$ is such that $\mathcal{G}_{\mathcal{H}} \supseteq \mathcal{F}$, the definition of tightness differs from the one given for sufficient conditions. Let $\mathcal{H}$, $\mathcal{H}'$ be two NFSCs, we say that $\mathcal{H}$ is *tighter* than $\mathcal{H}'$ if $\mathcal{G}_{\mathcal{H}} \subset \mathcal{G}_{\mathcal{H}'}$. As we have done for sufficient conditions, we first present a method that finds single graph NFSCs and then generalize this method to multiple graph NFSCs.

4.1 Single graph NFSC

The task of finding a single graph NFSC of a graph class $\mathcal{F}$ corresponds to finding a graph H such that, for all $G \in \mathcal{F}$, $H \not\subseteq G$. Thus, H is a "*common non-subgraph*" of the graphs in $\mathcal{F}$. Moreover, let H and H' be two graphs such

that $H \subset H'$. From Property 1d, we have that $\mathcal{G}_{\{H\}} \subset \mathcal{G}_{\{H'\}}$ and, therefore, that $\{H\}$ is tighter than $\{H'\}$. Thus, to maximize the tightness, H should be a common non-subgraph which is minimal w.r.t. inclusion. Our single graph NFSC algorithm, detailed in Figure 4, is based on this idea. Let $\overline{\mathcal{G}}_{\mathcal{H}} = \mathcal{G} \setminus \mathcal{G}_{\mathcal{H}}$ be the set of graphs containing at least one graph of $\mathcal{H}$, and denote by minCNS($\mathcal{L}$) the set of common non-subgraphs of a set $\mathcal{L}$, which have a minimum number of vertices. Starting with a graph H_0 that has a single vertex and an empty set $\mathcal{L}_0$ of representative graphs of $\mathcal{F}$, at each iteration k, we try to find a graph $G_k \in \mathcal{F}$ which contains H_k. If such a graph exists, then H_k is not an NFSC. We then improve H_k by adding G_k to the set of representatives of $\mathcal{F}$, i.e. $\mathcal{L}_{k+1} := \mathcal{L}_k \cup \{G_k\}$, and finding a graph H_{k+1} that is not a subgraph of any graph of $\mathcal{L}_{k+1}$. Finally, we repeat this process until no graph of $\mathcal{F}$ contains H_k, i.e. when $\mathcal{F} \cap \overline{\mathcal{G}}_{\{H_k\}} = \emptyset$, and return $\mathcal{H} = \{H_k\}$. In the case where $\mathcal{F}$ has no NFSC, we may always be able to find $G_k \in \mathcal{F}$ containing H_k, and thus, the algorithm may never stop. To avoid this problem, we impose a limit N on the number of vertices of our NFSC.

Single graph NFSC algorithm

Input: A graph class $\mathcal{F}$ and an integer $N > 0$.
Output: A single graph NFSC $\mathcal{H}$ of $\mathcal{F}$ having at most N vertices, or a graph of more than N vertices.

Let H_0 be a single vertex graph, $\mathcal{L}_0 := \emptyset$, and $k := 0$;

while $\exists G_k \in \mathcal{F} \cap \overline{\mathcal{G}}_{\{H_k\}}$ *and* $|V(H_k)| \leq N$ **do**
  Let $\mathcal{L}_{k+1} := \mathcal{L}_k \cup \{G_k\}$;
  Choose H_{k+1} in minCNS($\mathcal{L}_{k+1}$) ;
  Let $k := k + 1$;
return $\mathcal{H} = \{H_k\}$;

Figure 4: Algorithm to find an NFSC containing a single forbidden subgraph.

Proposition 5. *Let $\mathcal{F}$ be a graph class such that $\mathcal{F}$ is finite. The single graph NFSC algorithm of Figure 4 returns, in a finite number of steps, a common non-subgraph of $\mathcal{F}$ which is minimal w.r.t. inclusion and has at most N vertices, if one exists. Otherwise, the algorithm returns a graph that has more than N vertices.*

Proof. We first prove that the algorithm terminates in a finite number of steps. Consider any two graphs H_i and H_j such that $i < j$. Since H_j is a common non-subgraph of $\mathcal{L}_j$, and $G_i \in \mathcal{L}_j$, we have $H_j \not\subseteq G_i$ (*1*). Moreover, since $G_i \in \mathcal{F} \cap \overline{\mathcal{G}}_{\{H_i\}}$, we know that $H_i \subseteq G_i$ (*2*). Combining (*1*) and (*2*), we get $H_i \neq H_j$. Furthermore, since H_i and H_j are minimum common non-subgraphs of $\mathcal{L}_i$ and $\mathcal{L}_j$, and since $\mathcal{L}_i \subset \mathcal{L}_j$, we have that $|V(H_j)| \geq |V(H_i)|$. Finally, since there is a finite number of graphs of N or less vertices, the algorithm will terminate in a finite number of steps. Next, we prove that, if the graph H_k returned by the algorithm is such that $|V(H_k)| \leq N$, then it is a common non-subgraph of $\mathcal{F}$. Suppose it is not. Then, there exists a graph $G_k \in \mathcal{F}$ that has H_k as subgraph. However, this is a contradiction since the algorithm terminates only when no such graph exists, or if $|V(H_k)| > N$. Finally, we show that H_k is minimal w.r.t. inclusion. Suppose this is not the case. Then, there exists

a common non-subgraph H' of $\mathcal{F}$ such that $H' \subset H_k$. Since H' is a common non-subgraph of $\mathcal{F}$, it is also a common non-subgraph of $\mathcal{L}_k \subseteq \mathcal{F}$. However, this contradicts the fact that H_k is a minimum common non-subgraph of $\mathcal{L}_k$. $\square$

Although the algorithm is guaranteed to find in a finite number of steps a single graph NFSC, if one exists, we can accelerate its convergence using the fact that all NFSCs $\mathcal{H}$ are such that $\mathcal{H} \subseteq \overline{\mathcal{F}}$. Thus, instead of starting with a graph H_0 composed of a single vertex, we can start with H_0 as a minimum order graph of $\overline{\mathcal{F}}$. Furthermore, let $\text{minCNS}(\mathcal{L}, \mathcal{C})$ be the set containing the smallest graph of a class $\mathcal{C}$ that are not contained in any graph of $\mathcal{L}$, we can choose H_{k+1} in $\text{minCNS}(\mathcal{L}, \overline{\mathcal{F}})$ instead of $\text{minCNS}(\mathcal{L})$.

4.2 Multiple graph NFSC

As it was the case for sufficient conditions, we can sometimes improve the tightness of an NFSC by reducing its simplicity.

Proposition 6. *Let $\mathcal{F}$ be a given graph class and let $\mathcal{N}$ be the set of single graph NFSCs, i.e. $\mathcal{N} = \{H \in \overline{\mathcal{F}} \mid \nexists G \in \mathcal{F} \text{ s.t. } H \subset G\}$. If $\mathcal{H}$ is a minimal NFSC of maximum tightness then $\mathcal{H} = \min(\mathcal{N})$.*

Proof. Let $\mathcal{H} = \min(\mathcal{N})$. We first show that $\mathcal{H}$ is an NFSC of $\mathcal{F}$. Suppose it is not, then there is a graph $G \in \mathcal{F} \setminus \mathcal{H}$ such that $\exists H \in \mathcal{H}$, $H \subset G$, which contradicts the definition of $\mathcal{H}$. We next show that $\mathcal{H}$ is the only NFSC with maximum tightness. Let $\mathcal{H}'$ be any minimal NFSC of $\mathcal{F}$, and let H' be any graph of $\mathcal{H}'$. By definition, we know that $H' \in \mathcal{N}$. Suppose that H' is not in $\mathcal{H}$, since $\mathcal{H} = \min(\mathcal{N})$, there is a graph $H \in \mathcal{H}$ such that $H \subset H'$. Furthermore, from Properties 1d and 1e, we have that $\mathcal{G}_{\{H\}} \subset \mathcal{G}_{\{H'\}}$, and, thus, that the NFSC $\mathcal{H}''$ produced by substituting H' for H in $\mathcal{H}'$, i.e. $\mathcal{H}'' = (\mathcal{H}' \setminus \{H'\}) \cup \{H\}$, is tighter than $\mathcal{H}'$. Therefore, $\mathcal{H}'$ is maximally tight only if $\mathcal{H}' \subseteq \mathcal{H}$. Suppose finally that $\mathcal{H}' \subset \mathcal{H}$. Then, by Property 1f, we have that $\mathcal{G}_{\mathcal{H}'} \supset \mathcal{G}_{\mathcal{H}}$, and, thus, that $\mathcal{H}$ is tighter than $\mathcal{H}'$. Consequently, either $\mathcal{H}'$ does not have maximum tightness or $\mathcal{H}' = \mathcal{H}$. $\square$

Following the above proposition, the NFSC of a class $\mathcal{F}$ with maximum tightness $\mathcal{H}$ is simply the minimal set containing all individual graphs that are, by themselves, NFSCs. To find $\mathcal{H}$, we extend the previous algorithm such that it finds all the single graph NFSCs, as shown in Figure 5. We start with an empty set $\mathcal{M}_0$ of forbidden subgraphs, an empty set $\mathcal{L}_0$ of representative graphs of class $\mathcal{F}$, and a graph H_0 composed of a single vertex. We can also select H_0 as one of the smallest graph of $\overline{\mathcal{F}}$, to speed-up the convergence. Then, at each iteration k, we look for a graph $G_k \in \mathcal{F}$ that contains H_k. If such a graph exists, we add it to the set of representatives of $\mathcal{F}$, i.e. we set $\mathcal{L}_{k+1} := \mathcal{L}_k \cup \{G_k\}$. Otherwise, H_k is a single graph NFSC and we add it to the set of forbidden subgraphs, i.e. we set $\mathcal{M}_{k+1} := \mathcal{M}_k \cup \{H_k\}$. Then, we find the smallest graph H_{k+1} that is not contained in any graph of $\mathcal{L}_{k+1}$, nor contains any graph of $\mathcal{M}_{k+1}$, i.e. $H_{k+1} \in \text{minCNS}(\mathcal{L}_{k+1}, \mathcal{G}_{\mathcal{M}_{k+1}})$. Again, if we wish to speed-up the convergence of the algorithm, we can restrict our search to $\overline{\mathcal{F}}$. We repeat this process until the order of H_k exceeds a given limit N, and return $\mathcal{H} = \mathcal{M}_k$.

Multiple graph NFSC algorithm

Input: A graph class $\mathcal{F}$ and an integer $N > 0$.
Output: A set $\mathcal{H}$ containing all single graph NFSCs of at most N vertices.

Let H_0 be the graph containing a single vertex ;
Let $\mathcal{M}_0 := \emptyset$, $\mathcal{L}_0 := \emptyset$, and $k := 0$;

while $|V(H_k)| \leq N$ **do**
  if $\exists G_k \in \mathcal{F} \cap \overline{\mathcal{G}}_{\{H_k\}}$ **then** let $\mathcal{L}_{k+1} := \mathcal{L}_k \cup \{G_k\}$ and $\mathcal{M}_{k+1} := \mathcal{M}_k$;
  else let $\mathcal{L}_{k+1} := \mathcal{L}_k$ and $\mathcal{M}_{k+1} := \mathcal{M}_k \cup \{H_k\}$;
  Choose H_{k+1} in $\text{minCNS}(\mathcal{L}_{k+1}, \mathcal{G}_{\mathcal{M}_{k+1}})$;
  Let $k := k + 1$;
return $\mathcal{H} = \mathcal{M}_k$;

Figure 5: Algorithm to find a set containing all single graph NFSCs of at most N vertices.

Proposition 7. *Let $\mathcal{F}$ be a graph class such that $\mathcal{F}$ is finite. The multiple graph NFSC algorithm of Figure 5 produces, in a finite number of steps, a set $\mathcal{H}$ containing all single graph NFSCs of at most N vertices.*

Proof. We first prove that the algorithm terminates in a finite number of steps. At every iteration k, only one of the following three conditions is satisfied: (*1*) $|V(H_k)| > N$, (*2*) $\exists G_k \in \mathcal{F} \cap \overline{\mathcal{G}}_{\{H_k\}}$, or (*3*) $\nexists G_k \in \mathcal{F} \cap \overline{\mathcal{G}}_{\{H_k\}}$. If the condition (*2*) is verified, then, since $H_k \subseteq G_k$ and $H_{k+1} \not\subseteq G_k$, we have that $H_k \neq H_{k+1}$. Moreover, because H_k and H_{k+1} are minimum common non-subgraphs of $\mathcal{L}_k$ and $\mathcal{L}_{k+1}$, and since $\mathcal{L}_k \subset \mathcal{L}_{k+1}$, we have that $|V(H_{k+1})| \geq |V(H_k)|$. However, if condition (*3*) is verified, then we have $H_k \in \mathcal{M}_{k+1}$ and $H_{k+1} \in \mathcal{G}_{\mathcal{M}_{k+1}}$, and therefore $H_{k+1} \neq H_k$. Also, because H_k and H_{k+1} are minimum common non-subgraphs of $\mathcal{L}_k$ and $\mathcal{L}_{k+1}$, and since $\mathcal{L}_k = \mathcal{L}_{k+1}$, we have $|V(H_{k+1})| \geq |V(H_k)|$. Finally, since there is a finite number of graphs with N or less vertices, condition (*1*) will eventually be verified and the algorithm will terminate.

We next show that the algorithm is sound and complete. Since a graph H is only added to $\mathcal{H}$ if $\mathcal{F} \cap \overline{\mathcal{G}}_{\{H\}} = \emptyset$, we know that $\mathcal{F} \subseteq \mathcal{G}_{\{H\}}$ and thus that $\{H\}$ is an NFSC of $\mathcal{F}$. Furthermore, suppose there are two graphs $H_i, H_j \in \mathcal{H}$ such that $H_i \subset H_j$. Since, $|V(H_i)| < |V(H_j)|$, we know that $i < j$. Moreover, since $H_i \in \mathcal{M}_j$ and $H_j \in \mathcal{G}_{\mathcal{M}_j}$, we know that $H_i \not\subseteq H_j$ which is a contradiction. Therefore, the set $\mathcal{H}$ is minimal and the algorithm is sound. Moreover, suppose there exists a NFSC $\{H\}$ with $V(H) \leq N$, that is not found by the algorithm. By definition, we know that $H \in \text{minCNS}(\mathcal{F}, \mathcal{G}_{\mathcal{H}})$. Also, because minCNS finds the common non-subgraphs of minimum order, there must be an iteration k such that $H \notin \text{minCNS}(\mathcal{L}_k, \mathcal{G}_{\mathcal{M}_k})$. However, since $\mathcal{L}_k \subseteq \mathcal{F}$ and $\mathcal{M}_k \subseteq \mathcal{H}$, for every iteration k, we have $\text{minCNS}(\mathcal{F}, \mathcal{G}_{\mathcal{H}}) \subseteq \text{minCNS}(\mathcal{L}_k, \mathcal{G}_{\mathcal{M}_k})$, and thus $H \notin \text{minCNS}(\mathcal{F}, \mathcal{G}_{\mathcal{H}})$, which contradicts our initial supposition. Therefore, the algorithm is complete. □

5 Necessary and sufficient conditions

Before we present a method that finds FSCs, we need to answer some important questions. Firstly, we want to know under what conditions does a graph class have an FSC. The following proposition gives a well known result on the existence of an FSC for a given class of graphs, see e.g. [14].

Proposition 8. *A graph class $\mathcal{F}$ has an FSC if and only if it is hereditary.*

Proof. Let $\mathcal{H}$ be an FSC of $\mathcal{F}$. For any $G \in \mathcal{F}$, we know that G contains no graph of $\mathcal{H}$. Furthermore, for any $G' \subset G$, from Property 1b, we have that G' does not contain any graph of $\mathcal{H}$. Therefore, $G' \in \mathcal{G}_{\mathcal{H}} = \mathcal{F}$ and $\mathcal{F}$ is hereditary. Suppose conversely that $\mathcal{F}$ is an hereditary class of graphs, and let G be a graph of $\mathcal{F}$. Since every subgraph of G belongs to $\mathcal{F}$, we know that no subgraph of G belongs to $\overline{\mathcal{F}}$, and thus that $G \in \mathcal{G}_{\overline{\mathcal{F}}}$. Therefore, we have that $\mathcal{F} \subseteq \mathcal{G}_{\overline{\mathcal{F}}}$ (*1*). Moreover, let H be any graph of $\mathcal{G}_{\overline{\mathcal{F}}}$. We know that $H \notin \overline{\mathcal{F}}$, and therefore that $H \in \mathcal{F}$. Consequently, we have that $\mathcal{G}_{\overline{\mathcal{F}}} \subseteq \mathcal{F}$ (*2*). Combining (*1*) and (*2*), we have $\mathcal{F} = \mathcal{G}_{\overline{\mathcal{F}}}$, and thus, $\overline{\mathcal{F}}$ is an FSC of $\mathcal{F}$. □

The next question is equally important: *does a graph class $\mathcal{F}$ have a unique minimal FSC and, if so, what is it ?* This question is answered by the following propositions.

Proposition 9. *Let $\mathcal{H}$ and $\mathcal{H}'$ be two minimal forbidden subgraph characterizations, then $\mathcal{G}_{\mathcal{H}} = \mathcal{G}_{\mathcal{H}'}$ if and only if $\mathcal{H} = \mathcal{H}'$.*

Proof. Since $\mathcal{H} = \mathcal{H}' \Rightarrow \mathcal{G}_{\mathcal{H}} = \mathcal{G}_{\mathcal{H}'}$ is trivial, we only show that $\mathcal{G}_{\mathcal{H}} = \mathcal{G}_{\mathcal{H}'} \Rightarrow \mathcal{H} = \mathcal{H}'$. Suppose that $\mathcal{H} \neq \mathcal{H}'$, then there exists $H \in \mathcal{H}$ such that $H \notin \mathcal{H}'$, or $H' \in \mathcal{H}'$ such that $H' \notin \mathcal{H}$. Without loss of generality, suppose the first case in true. If there is no $H' \in \mathcal{H}'$ such that $H' \subset H$ then, for all $H'' \in \mathcal{H}'$, we know that $H \notin \mathcal{G}_{\{H\}}$ and that $H \in \mathcal{G}_{\{H''\}}$. Finally, from Property 1e, we have that $H \notin \mathcal{G}_{\mathcal{H}}$ and $H \in \mathcal{G}_{\mathcal{H}'}$, and therefore that $\mathcal{G}_{\mathcal{H}} \neq \mathcal{G}_{\mathcal{H}'}$. Otherwise, let H' be a graph in $\mathcal{H}'$ such that $H' \subset H$. There is no $H'' \in \mathcal{H}$ such that $H'' \subseteq H'$, otherwise we would have $H'' \subset H$ which contradicts the minimality of $\mathcal{H}$. Therefore, we have that $H' \notin \mathcal{G}_{\{H'\}}$ and that $H' \in \mathcal{G}_{\{H''\}}$, for all $H'' \in \mathcal{H}$. Finally, from Property 1e, we have that $H' \notin \mathcal{G}_{\mathcal{H}'}$ and $H' \in \mathcal{G}_{\mathcal{H}}$ and therefore that $\mathcal{G}_{\mathcal{H}} \neq \mathcal{G}_{\mathcal{H}'}$. □

Proposition 10. *Let $\mathcal{F}$ be a class of graphs. If $\mathcal{F}$ has an FSC, then the unique minimal FSC of $\mathcal{F}$ is* $\min(\overline{\mathcal{F}})$.

Proof. If $\mathcal{F}$ has an FSC, we know from Proposition 8 that it is hereditary and that $\overline{\mathcal{F}}$ is an FSC of $\mathcal{F}$. Moreover, by definition, we know that $\min(\overline{\mathcal{F}})$ is also an FSC of $\mathcal{F}$ that is minimal. Finally, we know from Proposition 9 that $\mathcal{F}$ has a single minimal characterization, which can only be $\min(\overline{\mathcal{F}})$. □

Consider an hereditary class of graphs $\mathcal{F}$ and let G be a graph in $\overline{\mathcal{F}}$. Furthermore, let $G' = G - \{v\}$ be a graph produced by removing from G any vertex $v \in V(G)$ and its incident edges. If $G' \in \mathcal{F}$, since $\mathcal{F}$ is hereditary, all subgraphs of G' are also in $\mathcal{F}$. Therefore, we know that a graph G is minimal in $\overline{\mathcal{F}}$ iff all its subgraphs produced by removing a vertex are in $\mathcal{F}$. This is the main principle behind our algorithm to find FSCs, shown in Figure 6. At each iteration k, we find a graph $G_k \in \overline{\mathcal{F}} \cap \mathcal{G}_{\mathcal{M}_k}$, where $\mathcal{M}_0 = \emptyset$. Then, we search for a vertex v of

G_k such that $G_k - \{v\}$, is still in $\overline{\mathcal{F}}$. If no such vertex exists, then G_k is minimal and we add it to $\mathcal{M}_k$. Otherwise, we remove v from G_k and look for another vertex with the same properties. We repeat this process until $\overline{\mathcal{F}} \cap \mathcal{G}_{\mathcal{M}_k} = \emptyset$, and return the set $\mathcal{H} = \mathcal{M}_k$. However, since an FSC can have an infinite number of graphs, we need to limit the size of the FSC to guarantee that the algorithm terminates in a finite number of steps. This limit is given as the parameter M of the algorithm.

FSC algorithm

Input: A graph class $\mathcal{F}$ and an integer $M > 0$.
Output: An FSC $\mathcal{H}$ of $\mathcal{F}$ having at most M graphs, or a set containing more than M graphs.

Let $\mathcal{M}_0 := \emptyset$ and $k := 0$;

while $\exists G_k \in \overline{\mathcal{F}} \cap \mathcal{G}_{\mathcal{M}_k}$ *and* $k \leq M$ **do**
  while $\exists v \in V(G_k)$ *s.t.* $G_k - \{v\} \in \overline{\mathcal{F}}$ **do** $G_k := G_k - \{v\}$;
  Let $\mathcal{M}_{k+1} := \mathcal{M}_k \cup \{G_k\}$;
  Let $k := k + 1$;
return $\mathcal{H} = \mathcal{M}_k$;

Figure 6: Algorithm to find an FSC of at most M forbidden subgraphs.

Proposition 11. *Let $\mathcal{F}$ be a graph class such that $\overline{\mathcal{F}}$ is finite. The FSC algorithm of Figure 6 returns, in a finite number of steps, the set $\mathcal{H} = \min(\overline{\mathcal{F}})$, if $|\min(\overline{\mathcal{F}})| \leq M$, or a set $\mathcal{H}$ containing $M + 1$ graphs of $\min(\overline{\mathcal{F}})$.*

Proof. Let G_i and G_j be two graphs, such that $i < j$. Since $G_i \in \mathcal{M}_j$ and $G_j \in \mathcal{G}_{\mathcal{M}_j}$, we have that $G_i \not\subseteq G_j$ and therefore that $G_i \neq G_j$. Moreover, since, at every iteration k, we have $|\mathcal{M}_k| = k$, the algorithm will terminate in at most $M + 1$ iterations. Furthermore, let G_k be any graph of $\mathcal{H}$. Suppose that G_k is not minimal, then there is a vertex $v \in V(G_k)$ such that $G_k - \{v\} \in \overline{\mathcal{F}}$, which is a contradiction since G_k is added to $\mathcal{H}$ only if no such vertex exists. Therefore, $\mathcal{H} \subseteq \min(\overline{\mathcal{F}})$. Finally, we know that the algorithm terminates only when $\overline{\mathcal{F}} \cap \mathcal{G}_{\mathcal{H}} = \emptyset$ or $|\mathcal{H}| = M + 1$. In the first case, we know that $\mathcal{H}$ is an FSC of $\mathcal{F}$, and since it is minimal, we know from Proposition 9 that it is unique, and thus $\mathcal{H} = \min(\overline{\mathcal{F}})$. Otherwise, $\mathcal{H}$ contains $M + 1$ graphs of $\min(\overline{\mathcal{F}})$. □

6 Automated conjecture generation

In the last sections, we presented some exact methods to find FSCs of a class of graphs $\mathcal{F}$, or conditions to have a characterization. Although these methods are guaranteed to yield finite FSCs, SFSCs or NFSCs of $\mathcal{F}$, if they exist, this might be impossible in practice. In many of our algorithms, we need to determine whether or not the sets $\mathcal{F} \cap \overline{\mathcal{G}}_{\mathcal{H}}$ and $\overline{\mathcal{F}} \cap \mathcal{G}_{\mathcal{H}}$ are empty, where $\mathcal{H}$ is a non-empty set of graphs. Since $\mathcal{F}$ and $\overline{\mathcal{F}}$ can be infinite, this may very well be an undecidable problem. To overcome this problem, we impose some restrictions on the graphs considered by our algorithms. Let $\mathcal{G}^n$ denote the graphs of $\mathcal{G}$ that have at most n vertices. We limit the graphs considered by our algorithms to $\mathcal{G}^n$, and use $\mathcal{G}^n \cap \mathcal{F}$ and $\mathcal{G}^n \cap \overline{\mathcal{F}}$ instead of $\mathcal{F}$ and $\overline{\mathcal{F}}$. Consequently, the guarantee

that the results obtained by our algorithm are FSCs or conditions to have an FSC no longer holds. However, we can still use these algorithms to generate conjectures based on the hypothesis that if these results are true for graphs of n or less vertices, then they must be true for all graphs.

6.1 Infinite and parametrized characterizations

While limiting the order of graphs considered by our algorithms makes the conjecture generation process practicable, it also limits the characterizations that can be found to characterizations containing a limited number of forbidden subgraphs, each of which have a finite number of vertices. Yet, characterizations containing an infinite number of forbidden subgraphs with unlimited vertices are common in graph theory. For instance, as mentioned before, the class of perfect graphs is characterized by an infinite set of forbidden subgraphs, composed of odd cycles of five or more vertices and their complements.

Interestingly, our algorithms can still be used to discover such infinite characterizations. Let $\mathcal{H}$ be the set of forbidden subgraphs returned by one of the algorithms for a class $\mathcal{F}$. The first step is to identify a common class (i.e., a common property) $\mathcal{C}$ of several subgraphs of $\mathcal{H}$. Note that these subgraphs may have several common properties, and the intervention of a human may be required to select the most relevant one. If such a class is found, we then extend the forbidden subgraphs characterizing $\mathcal{F}$ to all the graphs of $\mathcal{C}$. We illustrate this technique in the experimental section to reproduce the characterization of perfect graphs.

A similar technique can be used to find parametrized characterizations of a class $\mathcal{F}$. Let $\mathcal{C}$ be a class whose graphs are uniquely determined by a set of parameters $\{p_1, \ldots, p_m\}$, and denote by $\mathcal{C}(p_1, \ldots, p_m)$, the unique graph of $\mathcal{C}$ with parameters $p_1, \ldots, p_m$. For example, $K(p_1)$ denotes the clique of order p_1, and $K(p_1, p_2)$ is the complete bipartite graph whose partitions contain p_1 and p_2 vertices. The idea is to launch the conjecture generation process using different values of a given parameter t, e.g. the maximum order of graphs, and observe how the forbidden subgraphs we obtain vary according to t. If, for each value of t, we obtain a forbidden subgraph of $\mathcal{C}$ whose parameters are functions of t, i.e. if $p_1 = f_1(t), \ldots, p_m = f_m(t)$, we can then conjecture that the graphs of $\mathcal{F}$ with parameter t are characterized by $\mathcal{C}\left(f_1(t), \ldots, f_m(t)\right)$. We use this technique in the experimental section to find NFSCs parametrized by the maximum order of graphs.

6.2 Exploration approaches

The methods we have presented require to find graphs of a given class $\mathcal{F}$ that have at most n vertices. In the next sections, we present two approaches for this task: an enumerative approach and a heuristic one.

6.2.1 Enumerative approach

In order for the conjecture generation methods to be generic, we need a mechanism to input any class $\mathcal{F}$ in these methods, such that every graph of $\mathcal{F}$ can be accessed. An efficient approach would be to input $\mathcal{F}$ as an algorithm that enumerates the graphs of $\mathcal{F}$ one by one. While efficient algorithms exist for

certain classes of graphs such as trees or cycles, such enumeration algorithms may not exist for other interesting classes that have links to hard combinatorial problems. Instead, our enumerative approach requires class $\mathcal{F}$ to be given in the form of a predicate which takes as input a graph G and returns true only if $G \in \mathcal{F}$. The only requirement is that the predicate should be able to determine if G is in $\mathcal{F}$ in a finite number of steps, which implies that this problem needs to be decidable. Graphs are found using a generate and test approach, where we generate every graph G up to order n and test the membership of G in $\mathcal{F}$ with the predicate.

To enumerate the graphs of n or less vertices, we use the method proposed by McKay [19]. This method supposes that the vertices of a graph have a distinct index, and enumerates the set of such *labeled* graphs in a way that avoids generating two isomorphic graphs. Let $G = (V, E)$ be a labeled graph of order n. We denote by Γ_G the set containing the permutations on the vertices of G, i.e. the symmetric group of V. Moreover, given a permutation $\varphi \in \Gamma_G$, we write G^φ the graph permuted by φ, i.e. the graph with vertex set $V^\varphi = V$ and edge set $E^\varphi = \{(u, v) \mid \exists (x, y) \in E \text{ s.t. } u = \varphi(x) \text{ and } v = \varphi(y)\}$. The automorphism group of G is the set containing the automorphisms of G, i.e. the set $Aut(G) = \{\varphi \in \Gamma_G \mid G^\varphi = G\}$. Furthermore, the orbits of a vertex $v \in V$, denoted by $Orb(v)$ is the set of vertices u such that there exists an automorphism mapping v to u, i.e., $Orb(v) = \{u \in V \mid \exists \varphi \in Aut(G) \text{ s.t. } u = \varphi(v)\}$.

For each labeled graph G, we assign a unique string $code(G)$ produced by concatenating the elements of the upper half of the adjacency matrix A of G:

$$code(G) = (a_{1,2}\ a_{1,3}\ a_{2,3}\ \ldots\ a_{i,j}\ a_{i,j+1}\ \ldots\ a_{n-1,n})\,.$$

The canonical representation of G, written $canon(G)$, is then the lexicographically minimum string, under any permutation, i.e. $canon(G) = \min_{\varphi \in \Gamma_G} code(G^\varphi)$. Let φ^* be any permutation leading to the canonical representation of G, we use φ^* to define a partial order on the vertices of V: a vertex $u \in V$ precedes a vertex $v \in V$ if and only if $Orb(u) \neq Orb(v)$ and u precedes v in φ^*. This partial ordering is then used to transform the search space into a rooted tree with a function p that uniquely maps G to a parent graph $p(G)$ obtained by removing from G a minimum vertex according to the partial ordering. By traversing this tree in a depth-first manner, we can then explore every graph without visiting isomorphic copies.

An important benefit of this approach is that it allows one to efficiently prune the search space. To illustrate this, suppose we want to find a graph of $\mathcal{G}_\mathcal{H}$, and let G be any graph. If $G \notin \mathcal{G}_\mathcal{H}$, then there is a graph $H \in \mathcal{H}$ such that $H \subseteq G$. Since all supergraphs[1] of G will also contain H, we can avoid the exploration of the subtree rooted at G without fear of missing any graphs of $\mathcal{G}_\mathcal{H}$.

Complexity analysis The time complexity of the enumerative approach depends on three tasks: (*1*) generating all graphs up to order n, (*2*) testing the membership of a generated graph G in $\mathcal{F}$, and (*3*) checking if G contains any graph H of a set $\mathcal{H}$ as a subgraph.

In the generation process, the canonical representation of each explored graph G must be computed. Since two graphs are isomorphic if and only if

[1] A graph G is a supergraph of a graph H if and only if H is a subgraph of G.

they have the same canonical representation, finding the canonical representation of G is as difficult as solving the graph isomorphism problem, for which no general polynomial time algorithm is known [11]. In practice, however, it can be done very efficiently using graph invariants, such as done in [18]. Thus, instead of considering every permutation of Γ_G, when searching for the canonical representation, we can calculate the value of a powerful yet computationally efficient invariant on every vertex of G, and then consider only the permutations of vertices that have non-decreasing values of the invariant. Since most graphs exhibit little symmetry, only a few vertices have the same invariant value, and, in the general case, the canonical representation can be obtained quite fast. Furthermore, even though no polynomial time algorithm exists for computing the canonical representation, it can be shown that the generation process has polynomial delay [13]. That is, there are at most $P(n)$ steps between the generation of each non-isomorphic graph, where $P(n)$ is a polynomial on n.

The complexity of testing the membership of a graph G in $\mathcal{F}$, however, greatly depends on $\mathcal{F}$. For instance, testing whether a graph G is a perfect graph can be done in polynomial time through the detection of forbidden subgraphs [7]. However, using the predicate of Equation 1 requires computing the chromatic number and clique number of several subgraphs, two *NP*-hard problems [12]. On the other hand, determining if G is a split graph [10, 15], i.e. a graph in which the vertices can be partitioned into a clique and an independent set[2], can be done in $O(n \log n)$ time. Finally, the complexity of checking if G contains any graph of a set $\mathcal{H}$ as subgraph amounts to solving the NP-complete subgraph isomorphism problem [12] for each graph of $\mathcal{H}$.

Although testing the membership of a graph G in a given class and determining if G contains any graph of a set $\mathcal{H}$ as subgraph can both be expensive problems, the main limitation of the enumerative approach comes from the great number of graphs to explore. Thus, as shown in Table 1, there are more than 40 times as many graphs with 10 vertices than with 9 vertices, and too many graphs with 11 vertices or more to perform hard tasks such as testing the membership of a graph G in $\mathcal{F}$ and the inclusion of a graph of $\mathcal{H}$ in G. Therefore, this approach is more suitable when limiting the order of the explored graphs to ten vertices or less.

n	Graphs of order n
7	1,044
8	12,346
9	274,668
10	12,005,168
11	1,018,997,864
12	165,091,172,592

Table 1: Number of graphs on n vertices.

Tables 2 and 3 give some computational statistics illustrating the process of finding an FSC for the classes of perfect and split graphs. For n between 7 and 10, these tables give the CPU time in seconds required to find an FSC of at most n vertices, using a 2.2 GHz Intel dual core processor with 2Gb of RAM. For each run, these tables also present the number of membership in $\overline{\mathcal{F}}$

[2] A set W of vertices is independent if no two vertices of W are adjacent.

tests performed, and the number of graphs that have passed these tests, i.e. the explored graphs that actually belong to $\overline{\mathcal{F}}$. Finally, the tables show the total number of subgraph isomorphism tests performed in each run.

As expected, these statistics show an exponential increase in the CPU time, the number of $\overline{\mathcal{F}}$ membership tests, and the number of subgraph isomorphism tests performed for increasing values of n, due to the combinatorial explosion on the number of graphs to explore. It is interesting to note, however, that finding an FSC for the class of split graphs takes longer than for perfect graphs, even though split graphs are identifiable in $O(n \log n)$ time, while identifying perfect graphs is *NP*-hard. This result is due to the fact that, for $n \leq 10$, a small proportion of graphs are not perfect, while the majority of graphs are not split graphs. Since we test if a graph is $\mathcal{H}$-free only if it is in $\overline{\mathcal{F}}$, far more subgraph isomorphism tests are performed for the class of split graphs, as shown in Tables 2 and 3.

n	CPU (sec)	$\overline{\mathcal{F}}$ tests	$\overline{\mathcal{F}}$ graphs	Subgraph iso.
7	0.1	1,302	44	43
8	0.8	14,899	628	1,754
9	19.9	289,567	17,754	52,044
10	1381.5	12,583,000	1,028,874	4,996,063

Table 2: Computational statistics for the class of *perfect* graphs.

n	CPU (sec)	$\overline{\mathcal{F}}$ tests	$\overline{\mathcal{F}}$ graphs	Subgraph iso.
7	0.1	1,319	1,012	2,147
8	1.0	13,665	12,801	23,903
9	24.5	288,333	285,246	444,536
10	1113.8	12,293,501	12,279,648	15,864,305

Table 3: Computational statistics for the class of *split* graphs.

6.2.2 Heuristic approach

In the conjecture generation process, some graphs may be too large to be explored using the enumerative approach. In such cases, it is necessary to trade the completeness of that approach to gain the ability of finding graphs with more vertices. This sacrifice is somewhat reasonable, since we already limit the search space to graphs with n or less vertices.

The heuristic approach we use is similar to the one employed by AGX to find extremal graphs [5]. In this approach, the task of finding a graph of a class $\mathcal{F}$ is modeled as an optimization problem where the objective function f determines the membership of a graph G in $\mathcal{F}$. For example, we can express this function such that $f(G) \leq 0$ if and only if G belongs to $\mathcal{F}$, and use any heuristic search method to minimize f until we find $f(G) \leq 0$. In AGX, the heuristic used is the *Variable Neighborhood Search* [20]. This heuristic starts with a random graph $G \in \mathcal{G}^n$ and explores the neighborhood containing the graphs that can be obtained from G by adding/removing a vertex or a limited number of edges. If no neighbor graph improving G is found, the neighborhood is then widened by adding an increasingly large random perturbation to G before exploring its

neighborhood, until a neighbor improving G is found, or until the maximum neighborhood size is attained or a given time limit is reached. Furthermore, if a neighbor improving G is found, the search is then recentered on this neighbor and the original neighborhood is restored. Using this approach, AGX was able to find extremal graphs of up to 50 vertices.

To illustrate this approach, consider a set of m graphs invariants $\{\gamma_1, \gamma_2, \ldots, \gamma_m\}$, and a class containing every graph G such that

$$a_1\gamma_1(G) \ + \ a_2\gamma_2(G) \ + \ \ldots \ + \ a_m\gamma_m(G) \ \leq \ 0,$$

where $a_i \in \mathbb{R}$, $1 \leq i \leq m$. Furthermore, let $\mathcal{H} = \{H_1, H_2, \ldots, H_p\}$ be a set of p forbidden subgraphs, and denote by $\mathrm{emb}(H, G)$ the number of different embeddings, i.e. subgraph isomorphisms, of H in G. The task of finding G in $\overline{\mathcal{F}} \cap \mathcal{G}_{\mathcal{H}}$ can be formulated as minimizing

$$f(G) \ = \ \sum_{i=1}^{m} a_i\gamma_i(G) \ + \ B\sum_{j=1}^{p} \mathrm{emb}(H_j, G),$$

where B is a constant chosen such that any non-zero value of the right sum dominates the value of the left sum. If we find G such that $f(G) \leq 0$, we know that G is in $\overline{\mathcal{F}} \cap \mathcal{G}_{\mathcal{H}}$.

In the heuristic approach, like the enumerative approach, the membership of a graph G in a class $\mathcal{F}$ must be tested, and it must verified if G has any graph of a set $\mathcal{H}$ as a subgraph. Thus, if we consider the maximum order n as a parameter, this approach also runs in time exponential to n. However, unlike the enumerative approach, this approach visits a constant number of graphs, fixed by a user supplied parameter, which allows it to explore graphs with much more vertices.

The main problem with the heuristic approach, however, comes from the non-smoothness of the cost function. In some cases, the landscape defined by the cost function can be flat around the current graph G, making it difficult to select a transformation within the neighborhood of G. A solution to this problem consists in using a secondary cost function, which would guide the exploration toward a potentially interesting portion of the search space.

7 Experimental results

In this section, we use our algorithms to find results related to the concepts of independence, domination and irredundance of graphs. Before presenting the results, we need to define these concepts.

An *independent* (or *stable*) set S is a set of pairwise non-adjacent vertices. The *independence number* of a graph G, written $\alpha(G)$, is the maximum cardinality of an independent set of G, and the *independent domination number* of G, denoted by $i(G)$, is the minimum cardinality of a maximal independent set of G. Furthermore, a *dominating set* T is a set of vertices such that each vertex of $V(G) \setminus T$ is adjacent to at least one vertex of T. The *domination number* of G, written $\gamma(G)$, is the minimum cardinality of a dominating set of G, and we denote by $\Gamma(G)$ the maximum cardinality of a minimal dominating set of G. Moreover, let $X \subseteq V$, a vertex $x \in X$ is *irredundant* in X if it is isolated in X or if it has a private neighbor, i.e. a vertex $y \in V \setminus X$ such that x is the only vertex

of X adjacent to y. The set X is *irredundant* if all its vertices are irredundant. We denote by $IR(G)$ and $ir(G)$, respectively, the maximum cardinality of an irredundant set of G and the minimum cardinality of a maximal irredundant set of G. A famous result known as the *domination chain*, e.g. [17], states that the following relations hold for all graphs $G \in \mathcal{G}$:

$$ir(G) \leq \gamma(G) \leq i(G) \leq \alpha(G) \leq \Gamma(G) \leq IR(G).$$

In the next two sections, we search for SFSCs and NFSCs on the classes of graphs that satisfy or not some of these relations at equality. This is a two step process. First, we use our algorithms to find conjectures. Working under the assumption that the forbidden subgraphs have at most 10 vertices, we use the enumerative approach to find specific graphs in these algorithms. We then demonstrate by hand the conjectures found by our algorithms.

7.1 Experiments on SFSC

As a first experiment, we set out to find SFSCs for the classes of graphs which satisfy at equality the relations of the domination chain. This task has been the subject of previous work by graph theorists. In [1], Allan and Laskar showed that if a graph G does not contain the first graph of Figure 7, known as the "*Claw*", as subgraph, then $\gamma(G) = i(G)$. Furthermore, in [9], Favaron, based on some earlier work by Bollobas and Cockayne [4], conjectured that if G does not contain as subgraph any of the graphs of Figure 8, then $ir(G) = \gamma(G)$. This conjecture was later proved by Puech in [21], and by Volkmann and Zverovich in [23]. Favaron also showed in [9] that if G does not contain the first nor the second graph of Figure 7, known as the "*Deer*", then $ir(G) = i(G)$, and that if G does not contain any graph of the same figure, then $\Gamma(G) = IR(G)$.

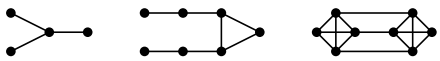


Figure 7: Graphs used in the SFSCs for graphs satisfying at equality some relations of the domination chain.

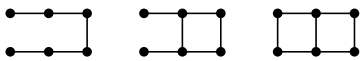
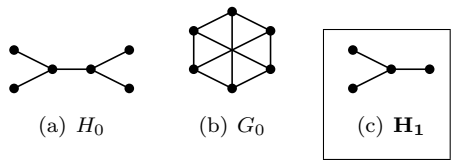


Figure 8: A SFSC for the class of graphs G such that $ir(G) = \gamma(G)$.

We first generated conjectures on SFSCs for the class of graphs G such that $\gamma(G) = i(G)$. Figure 9 shows a particular execution of our algorithm using $N = 1$ as maximum number of graphs and $\mathcal{G}^{10}$ as search space. At iteration $k = 0$, the algorithm finds a graph H_0 such that $\gamma(H_0) < i(H_0)$ and a graph G_0 that does not contain H_0 and such that $\gamma(G_0) < i(G_0)$. The algorithm then finds a graph H_1 that is a common subgraph of both H_0 and G_0. Since the algorithm cannot find an H_1-free graph G_1 such that $\gamma(G_1) < i(G_1)$, it returns $\mathcal{H} = \{H_1\}$, which is the SFSC proposed by Allan and Laskar in [1].

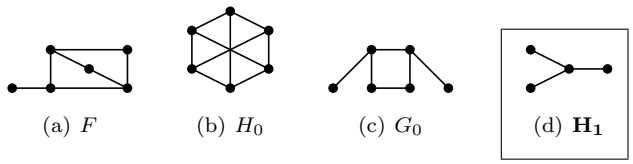
Note that we can obtain the same SFSC for higher values of N, as shown in Figure 10. In this example, where $N = 2$ is used, the algorithm first finds a



(a) H_0 (b) G_0 (c) $\mathbf{H_1}$

Figure 9: An illustration of the SFSC algorithm, for the class of graphs G such that $\gamma(G) = i(G)$, using $N = 1$ as maximum number of graphs and $\mathcal{G}^{10}$ as search space. Graph H_1 is an SFSC that was originally found by Allan and Laskar in [1].

graph $F \in \overline{\mathcal{F}}$ of minimum order, such that $\gamma(F) < i(F)$, and then searches for a maximum common subgraph of $\overline{\mathcal{F}} \cap G_{\{F\}}$. Since the algorithm cannot find a graph G_1 that does not contain F or H_1, and such that $\gamma(G_1) < i(G_1)$, it stops with $\{F, H_1\}$ as SFSC. However, since $H_1 \subset F$, the algorithm removes F from the SFSC and, once more, returns $\mathcal{H} = \{H_1\}$.



(a) F (b) H_0 (c) G_0 (d) $\mathbf{H_1}$

Figure 10: Another illustration of the SFSC algorithm, for the class of graphs G such that $\gamma(G) = i(G)$, using $N = 2$ as maximum number of graphs and $\mathcal{G}^{10}$ as search space. Graph H_1 is an SFSC that was originally found by Allan and Laskar in [1].

We have also used our algorithm to find conjectures on SFSCs for the class of graphs G such that $ir(G) = \gamma(G)$. In the execution shown in Figure 11, our algorithm generated the conjecture that the graphs F and H_3 of Figure 11, respectively known as the "*Deer*" and the "*Fork*", form an SFSC for this class of graphs. As we now prove, this conjecture is a novel result that strengthens Favaron's result, presented in [9], that $ir(G) = \gamma(G)$ if G is "*Claw*"-free and "*Deer*"-free.

Theorem 1. *If a graph G does not contain the graph F or H_3 of Figure 11, known respectively as the "*Deer*" and the "*Fork*", then $ir(G) = \gamma(G)$.*

Proof. Let I be a maximal irredundant set with $|I| = ir(G)$, and let A be the set of vertices in $V \setminus I$ that are not adjacent to any vertex of I. If A is empty then I is a dominating set and we have $\gamma(G) \leq ir(G)$. Thus, we assume that $A \neq \emptyset$ and denote by $N_X(v)$ the set of vertices in X that are adjacent to a vertex v. Furthermore, for a vertex $x \in I$, we denote by $P(x)$ the private neighbors of x, i.e. the vertices $y \in V \setminus I$ such that $N_I(y) = \{x\}$. Since I is irredundant, $P(x) \neq \emptyset$ for all $x \in I$ such that $N_I(x) \neq \emptyset$. Let $W \subseteq I$ be the subset of vertices of I redundant in $I \cup A$, implying by definition that $N_I(w) \neq \emptyset$ for all $w \in W$. For every $w \in W$, we denote by $A(w)$ the subset of vertices $a \in A$ such that w is redundant in $I \cup \{a\}$. Note that $\bigcup_{w \in W} A(w) = A$ and that $A(w) \neq \emptyset$ for every $w \in W$. Also, for each $w \in W$, every vertex $a \in A(w)$ is adjacent to every vertex $u \in P(w)$, or else w would still be irredundant in $I \cup \{a\}$, which

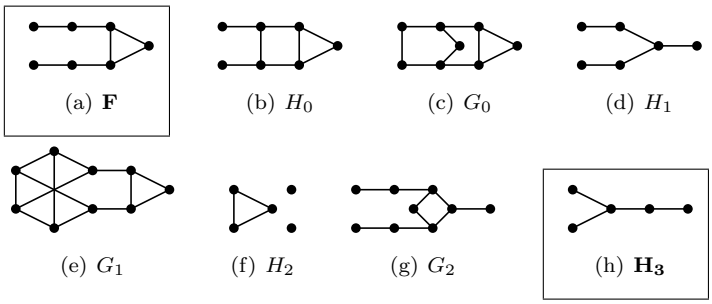


Figure 11: An illustration of the SFSC algorithm, for the class of graphs G such that $ir(G) = \gamma(G)$, using $N = 2$ as maximum number of graphs and $\mathcal{G}^{10}$ as search space. Graphs F and H_3 form a novel SFSC that strengthens a previous result proposed by Favaron in [9].

contradicts the definition of $A(w)$. Moreover, $P(w)$ forms a clique, otherwise there would be two non-adjacent vertices u_1 and u_2 in $P(w)$ such that $I \cup \{u_1\}$ is irredundant (since any vertex in $A(w)$ is a private neighbor of u_1 while u_2 is a private neighbor of w), a contradiction.

We now prove that every connected component of the subgraph $G[W]$ is a clique. This is obviously true for connected components with at most 2 vertices. So consider any connected component with at least three vertices, and assume it is not a clique. Then there are three vertices w_1, w_2 and w_3 in this component such that w_2 is adjacent to w_1 and w_3 while w_1 and w_3 are not adjacent. But then $\langle w_1, w_2, w_3, u, a\rangle \simeq H_3$ for all $u \in P(w_2)$ and $a \in A(w_2)$.

The next observation is that, for every $w \in W$, the vertices in $A(w)$ induce a clique. Indeed, suppose there are two non-adjacent vertices a_1 and a_2 in $A(w)$, and let u be any vertex in $P(w)$. Then, since $N_I(w) \neq \emptyset$, there is a vertex $x \in I$ adjacent to w, which means that $\langle w, x, u, a_1, a_2\rangle \simeq H_3$, a contradiction.

Note also that if C is the vertex set of a connected component of $G[W]$ with at least 3 vertices, then $\bigcup_{w\in C} A(w)$ induces a clique in G. Indeed, suppose this is not the case and consider two non-adjacent vertices a_1 and a_2 in $A(w_1) \cup A(w_2)$. Moreover, consider any $u_1 \in P(w_1)$ and $u_2 \in P(w_2)$. If both a_1 and a_2 are adjacent to u_1 or u_2 then they are adjacent to each other since $N_A(u_1)$ and $N_A(u_2)$ induce cliques, a contradiction. Otherwise, suppose, without loss of generality, that a_1 is adjacent only to u_1 and a_2 only to u_2. Then, vertices u_1 and u_2 are not adjacent, otherwise $\langle w_1, u_1, u_2, a_1, a_2\rangle \simeq H_3$. Furthermore, consider any vertex $w_3 \in C \setminus \{w_1, w_2\}$. As shown above, $\{w_1, w_2, w_3\}$ forms a clique and, thus, $\langle w_1, w_2, w_3, u_1, u_2, a_1, a_2\rangle \simeq F$, a contradiction.

Two connected components C and C' of $G[W]$ are said to be *I-linked* if there exists a vertex $x \in I$ with neighbors in both C and C'. Note that such a vertex x cannot belong to W since all connected components of $G[W]$ are cliques. Also, such a vertex x is necessarily adjacent to all vertices in C and C'. Indeed, let $w \in C$ and $w' \in C'$ be two neighbors of a vertex $x \in I$: if a vertex $w'' \in C$ (or $w'' \in C'$) is not adjacent to x, then $\langle w, w', w'', x, u\rangle \simeq H_3$ for any $u \in P(w)$ (or $u \in P(w')$). We now transform I into a set I' as follows. We first consider all connected components of $G[W]$ as unmarked. Then, as long as there exist two unmarked I-linked components C and C', we mark C and C', and do the following : if there is a vertex a that belongs to $A(w)$ for all $w \in C \cup C'$, we

choose one of the two components, say C, we replace a vertex $w \in C$ by a, and we replace every vertex w' of the other component C' by a vertex $u \in P(w')$; otherwise, we replace every vertex $w \in C \cup C'$ by a vertex in $P(w)$. Note that $|I'| \leq |I|$ (and $|I'|$ is possibly strictly smaller than $|I|$ since a vertex of A can replace several vertices of I).

Let W' denote the set of vertices in the unmarked connected components of $G[W]$. In what follows, all marked connected components of $G[W]$ are said to be of type 0, those of $G[W']$ with one vertex are said to be of type 1, and those of $G[W']$ with at least 3 vertices are said to be of type 2. We split the connected components of $G[W']$ with 2 vertices w_1, w_2 into three disjoint groups.

1. If there is a vertex in $P(w_1) \cup P(w_2)$ adjacent to all vertices in $A(w_1) \cup A(w_2)$, the component is said to be of type 3.

2. If every vertex in $P(w_1) \cup P(w_2)$ has a non-neighbor in $A(w_1) \cup A(w_2)$, and there is a vertex $a \in A(w_j)$ with a non-neighbor in $P(w_i)$ and adjacent to all vertices in $A(w_i)$ $(j \neq i)$, the component is said to be of type 4.

3. If every vertex in $P(w_1) \cup P(w_2)$ has a non-neighbor in $A(w_1) \cup A(w_2)$, and every vertex in $A(w_j)$ is either adjacent to all vertices in $P(w_i)$ or has a non-neighbor in $A(w_i)$ $(j \neq i)$, the component is said to be of type 5.

We now create a set I'' from I' as follows.

1. For every connected component $C = \{w\}$ of type 1, we replace w by a vertex $u \in P(w)$.
2. For every connected component C of type 2, we replace a vertex $w \in C$ by a vertex $a \in A(w)$.
3. For every connected component $C = \{w_1, w_2\}$ of type 3, there is a vertex $u \in P(w_j)$ adjacent to all vertices in $A(w_i)$ $(j \neq i)$. We replace w_j by u.
4. For every connected component $C = \{w_1, w_2\}$ of type 4, there is a vertex $a \in A(w_j)$ with a non-neighbor in $P(w_i)$ and adjacent to all vertices in $A(w_i)$ $(j \neq i)$. We replace w_j by a.
5. For every connected component $C = \{w_1, w_2\}$ of type 5, we replace w_1 and w_2 by a vertex $u_1 \in P(w_1)$ and a vertex $u_2 \in P(w_2)$.

Since $|I''| \leq |I'| \leq |I| = ir(G) \leq \gamma(G)$, it is now sufficient to prove that I'' is a dominating set. So let v be a vertex that does not belong to I''.

1. If $v \in I$ then v was removed from a connected component C. If C is of type 0, 2, 3 or 4, then there is at least one vertex adjacent to v that was not removed from I. Else, v is adjacent to the vertex in $P(v)$ that replaces v in I''.
2. If $v \in \bigcup_{w \in W} P(w)$ then let w be the vertex in W such that $v \in P(w)$. If $w \in I''$, then w is a neighbor of v in I''. Otherwise, w was removed from I and replaced by a vertex in $P(w) \cup A(w)$. It follows that v has a neighbor in I'' since $P(w)$ is a clique and v is adjacent to all the vertices in $A(w)$.
3. If $v \in A$ then let w be a vertex in W such that $v \in A(w)$ and let C be the connected component of $G[W]$ containing w.
   - If C is of type 0, then either exactly one vertex of C was replaced by a vertex in $A(w)$, or w was replaced by a vertex in $P(w)$. It follows that v has a neighbor in I'' since every vertex in $A(w)$ is adjacent to every vertex in $P(w)$, and $A(w)$ is a clique.

- If C is of type 1 or 3, then a vertex w' in C was replaced by a vertex $u \in P(w')$, and v has a neighbor in I'' since u is adjacent to all vertices in $\bigcup_{x \in C} A(x)$.
- If C is of type 2 or 4, then a vertex in C was replaced by a vertex $a \in \bigcup_{x \in C} A(x)$, and v has a neighbor in I'' since a is adjacent to all other vertices in $\bigcup_{x \in C} A(x)$.
- If C is of type 5, then its two vertices w_1 and w_2 were replaced by two vertices $u_1 \in P(w_1)$ and $u_2 \in P(w_2)$, which means that v has a neighbor in I'' since $v \in N_A(u_1) \cup N_A(u_2)$.

4. If $v \notin \left(I \cup A \cup \bigcup_{w \in W} P(w)\right)$ then v is adjacent to at least two vertices in I. From now on, we assume that v has no neighbor in I'', which implies $N_I(v) \subseteq W$.

   (a) If v has a neighbor in a connected component of type 0, then let C_1 and C_2 be two I-linked connected component that were marked together when constructing I', and such that $N_{C_1}(v) \cup N_{C_2}(v) \neq \emptyset$. Let x be a vertex in I adjacent to all vertices in $C_1 \cup C_2$ and let U denote the set of vertices in I' that replace vertices in $C_1 \cup C_2$ and are therefore not adjacent to v.

   - If no vertex a belongs to $\bigcap_{w \in C_1 \cup C_2} A(w)$, then consider any $w_1 \in C_1$ and any $w_2 \in C_2$ and let u_1 and u_2 be the two vertices in $(P(w_1) \cup P(w_2)) \cap U$. Note that u_1, u_2 and x belong to I' and are therefore not adjacent to v. Moreover, v is adjacent to either both or none of w_1 and w_2, otherwise $\langle w_1, w_2, x, u_1, v \rangle \simeq H_3$ or $\langle w_1, w_2, x, u_2, v \rangle \simeq H_3$. It follows that v is adjacent to all vertices in $C_1 \cup C_2$ (since v has at least one neighbor in $C_1 \cup C_2$). Observe now that every $a \in \bigcup_{w \in C_1 \cup C_2} A(w)$ is adjacent to either both or none of u_1 and u_2, otherwise $\langle w_1, x, u_1, v, a \rangle \simeq H_3$, or $\langle w_2, x, u_2, v, a \rangle \simeq H_3$. Hence, every vertex $a \in \bigcup_{w \in C_1 \cup C_2} A(w)$ is adjacent to every vertex in U (since every vertex in U has at least one neighbor in $\bigcup_{w \in C_1 \cup C_2} A(w)$), and every such vertex a is also adjacent to v, otherwise $\langle w_1, x, u_1, v, a \rangle \simeq H_3$.
     Consider now two vertices w and a such that $w \in C_1 \cup C_2$ and $a \in A \setminus A(w)$, and let u be a vertex in $P(w)$ not adjacent to a. Without loss of generality, we can assume $w \in C_1$. Let w' be any vertex in C_2 and let u' be the vertex in $P(w') \cap U$. We then have $\langle w, x, u, v, a \rangle \simeq H_3$ (if v is not adjacent to u) or $\langle w', x, u, v, a \rangle \simeq H_3$ (if v is adjacent to u), a contradiction.
   - If there is a vertex $a \in \bigcap_{w \in C_1 \cup C_2} A(w)$, then, without loss of generality, assume that a vertex $w_1 \in C_1$ was replaced by a. Consider any vertex $u_1 \in P(w_1)$, let w_2 be any vertex in C_2, and let u_2 be the vertex that replaces w_2 in I'. Observe that v is not adjacent to x, u_2, a and w_2 since $\{x, u_2, a\} \subseteq I'$ and $\langle w_2, x, u_2, v, a \rangle \simeq H_3$. Hence, v has no neighbor in C_2, and since all vertices in C_1, except w_1, belong to I', w_1 is the unique neighbor ot v in C_1. Let w_3 be a second neighbor of v in W (which implies $w_3 \notin C_1 \cup C_2$). Note that x is not adjacent to w_3, otherwise $\langle w_1, w_2, w_3, x, u_2 \rangle \simeq H_3$. Since I is irredundant and x has a neighbor in I, there is at least one vertex x' in $P(x)$, and such a

vertex is adjacent to v and u_2, otherwise $\langle w_1, w_2, x, x', v\rangle \simeq H_3$ or $\langle w_1, w_2, x, x', u_2\rangle \simeq H_3$. But then $\langle w_3, x, x', u_2, v\rangle \simeq H_3$, a contradiction.

(b) If v has a neighbor w in a connected component C of type 2, then let x be any vertex in $I \setminus \{w\}$ adjacent to v. Note that $x \notin C$ since $C \setminus \{w\} \subseteq I'$. Consider any two vertices w' and w'' in $C \setminus \{w\}$, and let u be any vertex in $P(w')$ and a any vertex in $A(w')$. Vertex v is not adjacent to u or a, otherwise $\langle w, w'', x, u, v\rangle \simeq H_3$ or $\langle w, w', x, v, a\rangle \simeq H_3$. Hence $\langle w, w', w'', x, u, v, a\rangle \simeq F$, a contradiction.

(c) If v has a neighbor w in a connected component C is of type 1, 3 or 5, then w was replaced by a vertex $u \in P(w)$. We first prove that if v has a neighbor in another connected component of $G[W]$, then v is not adjacent to any vertex in $A(w)$. So, assume there is a vertex $a \in A(w)$ adjacent to v. Since $I \cup \{v\}$ is redundant, while a is a private neighbor or v, there is a vertex $y \in I$ such that v is adjacent to all vertices in $P(y)$.

- If w is adjacent to y, then y belongs to a component C' which is not of type 0 (as shown in (a)) or 1 (since the vertex in $P(y)$ that replaces y in I'' whould be adjacent to v). Hence, y is adjacent to another vertex $z \in C'$ that belongs to I''. Since v is not adjacent to z, we have $\langle w, v, a, y, z\rangle \simeq H_3$, a contradiction.
- If w is not adjacent to y, then let w' be a neighbor of v in a connected component $C' \neq C$, and let u' be any vertex in $P(w')$. Vertex u is adjacent to all vertices $y' \in P(y)$, otherwise $\langle y', v, w', w, u\rangle \simeq H_3$, and y is adjacent to w', otherwise $\langle u, y', y, v, w'\rangle \simeq H_3$. Hence, y is adjacent to w, otherwise $\langle w, v, a, w', y\rangle \simeq H_3$. Therefore, C and C' are I-linked and not of type 0 (as shown in (a)), which means that the construction of I' from I was stopped too early, a contradiction.

Hence, if C is of type 1 or 3, then v is not adjacent to any vertex in $A(w)$, while if C is of type 5, v can be adjacent to a vertex in $A(w)$ only if the two vertices in C are the unique neighbors of v in I.

(c.1) If C is of type 1 or 3, then let a be any vertex in $A(w)$. Since $N_I(w) \neq \emptyset$, there is a vertex $w' \in I$ adjacent to w. Note that $w' \in I''$, since w is the unique vertex of C that was removed from I. Hence, v is not adjacent to w', which implies $\langle v, w, w', u, a\rangle \simeq H_3$, a contradiction.

(c.2) If C is of type 5, then let w' be the second vertex in C, and let $u' \in P(w')$ be the vertex that replaces w' in I'' (and is therefore not adjacent to v). Since no vertex in $P(w')$ is adjacent to all vertices in $A(w)$, consider a vertex $a \in A(w)$ not adjacent to u'. Since a is not adjacent to all vertices in $A(w')$, consider any $a' \in A(w')$ not adjacent to a. It follows that u is not adjacent to a' and u', otherwise $\langle w, w', u, a, a'\rangle \simeq H_3$, or $\langle w, u, u', a, a'\rangle \simeq H_3$. If v is adjacent to a then, as shown above, v is adjacent to w and w' and to no other vertex in I. Hence, v is not adjacent to a', otherwise $\langle w, u', v, a, a'\rangle \simeq H_3$. Since $I \cup \{v\}$ is redundant and a is a private neighbor of v, there is a vertex $y \in I$ such that v

is adjacent to all vertices in $P(y)$. Vertex y is not adjacent to w or w', otherwise $\langle w, y, u', v, a' \rangle \simeq H_3$ or $\langle w, w', y, u, u' \rangle \simeq H_3$. Consider any vertex $y' \in P(y)$. Vertex y' is not adjacent to u', otherwise $\langle u', y', y, v, w \rangle \simeq H_3$. Also, y' is adjacent to a but not to a', otherwise $\langle w, y, y', v, a \rangle \simeq H_3$ or $\langle y, y', u', a, a' \rangle \simeq H_3$. But then $\langle w, w', y, y', u', v, a' \rangle \simeq F$, a contradiction.
Hence, v is not adjacent to a, but is therefore adjacent to w', otherwise $\langle w, w', u, v, a \rangle \simeq H_3$. By permuting the roles of w, u, a with those of w', u', a', we also conclude that v is not adjacent to a', and we have $\langle v, w, w', u, u', a, a' \rangle \simeq F$, a contradiction.

(d) If v has a neighbor w in a connected component C of type 4, then let x be any vertex in $I \setminus \{w\}$ adjacent to v, and let $C' \neq C$ be the connected component of $G[W]$ containing x. We have shown above that if C' is of type 0, 1, 2, 3 or 5, then v has a neighbor in I''. So C' is also of type 4. Consider any vertex $u \in P(w)$, and let w' be the second vertex in C and x' the second vertex in C'. Vertex w was replaced by a vertex $a \in A(w)$ that is not adjacent to a vertex $u' \in P(w')$. We know that v is not adjacent to w', x', a (since these three vertices belong to I''), but is adjacent to u, otherwise $\langle w, w', u, v, a \rangle \simeq H_3$. Moreover, u' is not adjacent to v or u, otherwise $\langle w, x, x', u', v \rangle \simeq H_3$ or $\langle x, v, u, u', a \rangle \simeq H_3$. But then $\langle w, w', x, x', u, u', v \rangle \simeq F$, a contradiction.

□

If we combine the SFSCs we obtained for the classes $ir(G) = \gamma(G)$ and $\gamma(G) = i(G)$, we get the SFSC proposed by Favaron: if a graph G is "*Claw*"-free and "*Deer*"-free then $ir(G) = i(G)$. However, if the SFSC algorithm randomly selects the maximum common subgraphs, at each iteration, we can obtain a completely different SFSC. For instance, in the execution shown in Figure 12, our algorithm found a path of 5 vertices, which was shown by Puech in [21] to be an SFSC.

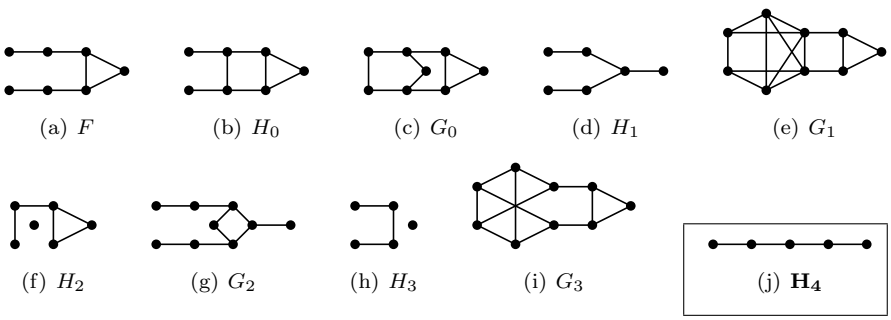


Figure 12: Another illustration of the SFSC algorithm, for the class of graphs G such that $ir(G) = \gamma(G)$, using $N = 2$ as maximum number of graphs and $\mathcal{G}^{10}$ as search space. Graph H_4 was shown be an SFSC by Puech in [21].

7.2 Experiments on NFSC

Following what we have done for SFSCs, we look in this section for NFSCs on the classes of graphs satisfying at inequality some of the relations of the domination chain.

As a first experiment, we used our algorithm to find conjectures on NFSCs for the class of graphs G such that $\gamma(G) < i(G)$. Figure 13 shows a particular execution of the algorithm for this task, using $\mathcal{G}^8$ as search space. At iteration $k = 0$, the algorithm finds a graph G_0 such that $\gamma(G_0) < i(G_0)$. Since $\mathcal{L}_1$ only contains G_0, the algorithm then finds a minimum order graph H_1 not included in G_0. At the next iteration, the algorithm then finds a graph G_1 containing H_1 and such that $\gamma(G_1) < i(G_1)$, and a graph H_2 of minimum order that is not a subgraph of G_0 or G_1. This process is repeated until the algorithm reaches iteration $k = 5$, where it finds a 5 vertex clique H_5, as the minimum order graph not included in $\{G_0, G_1, G_2, G_3, G_4\}$. However, the algorithm does not find a graph G_5 containing H_5 and such that $\gamma(G_5) < i(G_5)$, and thus returns $\mathcal{H} = \{H_5\}$. Repeating this experiment using $\mathcal{G}^9$ and $\mathcal{G}^{10}$ as search space, our algorithm has found complete graphs of 6 and 7 vertices. Following the technique presented in Section 6.1 to find parametrized characterizations, where we use the maximum order n as the parameter, we get the NFSC that a graph G of n vertices is $K_{(n-3)}$-free if $\gamma(G) < i(G)$. We now demonstrate this novel result.

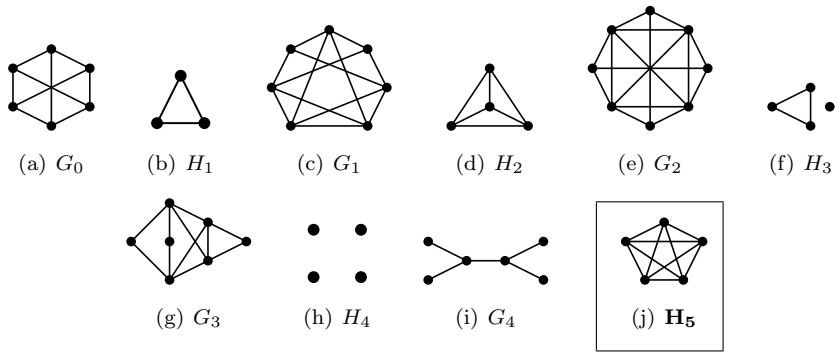


Figure 13: An illustration of the NFSC algorithm, for the class of graphs G such that $\gamma(G) < i(G)$, using $\mathcal{G}^8$ as search space.

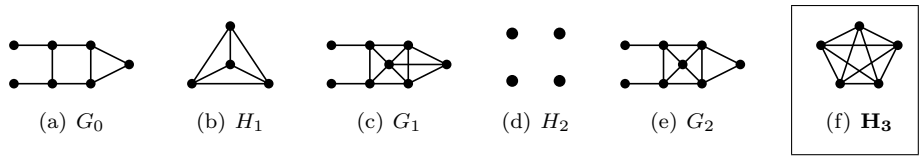
Theorem 2. *Let G be any graph of order n. If $\gamma(G) < i(G)$ then G is $K_{(n-3)}$-free.*

Proof. We will show that $i(G) \leq \gamma(G)$, if G contains a $K_{(n-3)}$. This will be done by showing that a minimum cardinality dominating set can be transformed into an independent dominating set of the same cardinality.

Let $D = \{w_1, w_2, \ldots, w_m\}$ be a minimum cardinality dominating set of G, i.e. $|D| = \gamma(G)$. If $E(G[D]) = \emptyset$, then D is an independent set, and we have $i(G) \leq |D| = \gamma(G)$. Otherwise, let (w_1, w_2) be any edge of $E(G[D])$, and let $P(w)$ be the private neighbors of w, i.e. the vertices $y \in V \setminus D$ such that $N_D(y) = \{x\}$. We know that $P(w_i) \neq \emptyset$, $i = 1, 2$, otherwise $D' = D \setminus \{w_i\}$ would be a dominating set of one less vertex than D, which contradicts the minimality of D. We next show that at least one of $P(w_1)$ or $P(w_2)$ is a complete

graph. Suppose this is not the case, then there are four vertices $u_1 \in P(w_1)$, $v_1 \in P(w_1)$, $u_2 \in P(w_2)$, $v_2 \in P(w_2)$ such that u_1, v_1 are non-adjacent, and u_2, v_2 are non-adjacent. However, $\langle u_1, v_1, w_1, u_2, v_2, w_2 \rangle$ contains no triangle, and thus G contains no $K_{(n-3)}$, which is a contradiction. Therefore, $P(w_1)$ or $P(w_2)$ forms a complete graph, without loss of generality, suppose it is $P(w_1)$. Let u_1 be any vertex of $P(w_1)$. The set $D' = (D \setminus \{w_1\}) \cup \{u_1\}$ is a dominating set of cardinality $|D'| = |D|$ such that $|E(G[D'])| < |E(G[D])|$. By repeating this process iteratively at most $n-3$ times, we will necessarily get an independent dominating set D'', such that $i(G) \leq |D''| = |D| = \gamma(G)$. □

In the next experiment, we searched for NFSCs on the class of graphs G such that $ir(G) < \gamma(G)$. Figure 14 shows an execution of our NFSC algorithm, using $\mathcal{G}^8$ as search space, where a complete graph of 5 vertices is found. Again, when repeating this experiment with G^9 and G^{10}, our algorithm found, although not systematically, complete graphs of 6 and 7 vertices, which suggested a new theorem and corollary:



(a) G_0 (b) H_1 (c) G_1 (d) H_2 (e) G_2 (f) $\mathbf{H_3}$

Figure 14: An illustration of the NFSC algorithm, for the class of graphs G such that $ir(G) < \gamma(G)$, using $\mathcal{G}^8$ as search space.

Theorem 3. *Let G be any graph of order n. If $ir(G) < \gamma(G)$ then G is $K_{(n-3)}$-free.*

Proof. Similar to what was done for the last theorem, we will prove that $\gamma(G) \leq ir(G)$ if G contains a $K_{(n-3)}$, by showing that a minimum cardinality maximal irredundant set can be transformed into a dominating set of the same cardinality.

Let I be a minimum cardinality maximal irredundant set of G, i.e. $|I| = ir(G)$, and let A be the set of vertices not dominated by I. If $A = \emptyset$, then I is a dominating set, and we have $\gamma(G) \leq |I| = ir(G)$. Otherwise, consider any vertex $a \in A$. There is a non-isolated vertex $w \in I$ which is redundant in $I \cup \{a\}$, otherwise I would not be maximal. Let $P(w)$ denote the private neighbors of w. We know that $P(w)$ is not empty since w is non-isolated in I. Furthermore, we know that a is adjacent to all the vertices of $P(w)$, otherwise w would be irredundant in $I \cup \{a\}$. We also know that $P(w)$ forms a complete graph, otherwise there are two vertices u_1, u_2 in $P(w)$ such that $I \cup \{u_1\}$ is irredundant (since u_1 has a as private neighbor and w the vertex u_2, thus contradicting the maximality of I). Denote by $A(w) \subseteq A$ the vertices a such that w is redundant in $I \cup \{a\}$, and let W be the vertices of I that are redundant in $I \cup A$.

Consider any vertex $w_1 \in W$ such that $|A(w_1)| = \max_{w \in W} |A(w)|$. We show that $A(w_1) = A$. Suppose this is not the case, then consider any vertex $a_2 \in A \setminus A(w_1)$, and let w_2 be any vertex redundant in $I \cup \{a_2\}$. Since a_2 belongs to $A(w_2)$ but not to $A(w_1)$, and since $|A(w_1)| \geq |A(w_2)|$, there is a vertex $a_1 \in A(w_1)$ that does not belong to $A(w_2)$. Let u_1 be a vertex in $P(w_1)$ that is not adjacent to a_2, and u_2 be a vertex in $P(w_2)$ that is not adjacent to

a_1. Then, $\langle a_1, u_1, w_1, a_2, u_2, w_2 \rangle$ contains no triangle, and thus G contains no $K_{(n-3)}$, a contradiction. Therefore, $A(w_1) = A$ and, since any vertex of $A(w_1)$ is adjacent to every vertex of $P(w_1)$ and because $P(w_1)$ forms a complete graph, the set $I' = (I \setminus \{w_1\}) \cup \{u_1\}$ is a dominating set of G. Thus, we have that $\gamma(G) \leq |I'| = |I| = ir(G)$. □

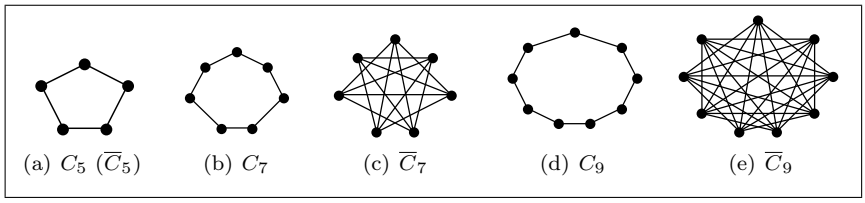
Corollary 1. *Let G be any graph of order n. If $ir(G) < i(G)$ then G is $K_{(n-3)}$-free.*

7.3 Experiments on FSC

In the broad category of graph classes that can be defined without a set of forbidden subgraphs, e.g. classes defined using invariants, those with the heredity property are not common. Moreover, known hereditary classes have been studied in depth and have already been characterized with forbidden subgraphs. While we could define new classes, for instance by generating equations on a set of graph invariants, these classes would most likely not be hereditary. Thus, rather than trying to discover new FSCs, we instead illustrate our FSC algorithm on well known hereditary classes: perfect graphs and split graphs. Still, if new hereditary classes are eventually described in the literature, this algorithm could help facilitate their characterization.

7.3.1 Perfect graphs

A graph G is perfect if, for each subgraph $G' \subseteq G$, the chromatic number $\chi(G')$ of G' is equal to its clique number $\omega(G')$. Because the clique number of any graph is never greater than its chromatic number, a graph G is *imperfect* if it has a subgraph $G' \subseteq G$ for which $\omega(G') < \chi(G')$. However, since we are only interested in finding *minimal* imperfect graphs, and because we search the graph space bottom-up, it is sufficient to test whether $\omega(G) < \chi(G)$. Using this test, and limiting the search space to $\mathcal{G}^{10}$, we found the five graphs shown in Figure 15. Knowing that the complement graph of the five vertex cycle C_5 is itself C_5, it can be observed (through human intervention) that the forbidden subgraphs found by the algorithm are either odd cycles or their complement. Following the idea presented in Section 6.1, we can extend the characterization to the infinite set composed of all odd cycles of five or more vertices and their complement, the famous result of Chudnovsky et al.[6].



(a) C_5 ($\overline{C}_5$) (b) C_7 (c) $\overline{C}_7$ (d) C_9 (e) $\overline{C}_9$

Figure 15: An illustration of the FSC algorithm for the class of perfect graphs, using $\mathcal{G}^{10}$ as search space.

7.3.2 Split graphs

Split graphs, i.e. graphs whose vertices can be partitioned into a clique and an independent set, possess some very interesting properties that have been studied, among others, by Földes, Hammer and Simeone [10, 15]. One of these properties is that they can be recognized in $O(n \log n)$ time, using the sequence of degrees of each vertex. Let G be a graph with n vertices, we order the degrees of the vertices of G, by non-increasing value, to form a sequence $d_1 \geq d_2 \geq \ldots \geq d_n$. Let m be the largest value of i such that $d_i \geq i - 1$, then G is a split graph if and only if

$$m(m-1) \; - \; \sum_{i=1}^{m} d_i \; + \; \sum_{j=m+1}^{n} d_j \; \leq \; 0.$$

If this is the case, the m vertices with the largest degrees form a maximum clique in G, and the remaining vertices an independent set.

Using the above test to determine whether or not a graph is a split graph and $\mathcal{G}^8$ as search space, we have found the minimal FSC for this class, shown in Figure 16, containing two cycles of 4 and 5 vertices, and a graph made of two disjoint edges. This FSC is a well known result, first described by Földes and Hammer in [10].

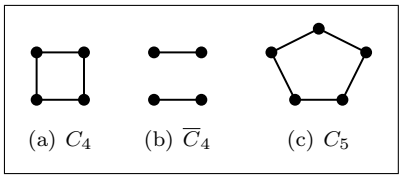


Figure 16: An illustration of the FSC algorithm for the class of split graphs, using $\mathcal{G}^8$ as search space.

8 Conclusion

We have presented, in this paper, methods to automate the discovery of new conjectures on FSCs. Although methods have already been proposed for the automated generation of conjectures, none of these methods have explored conjectures on FSCs.

We have first described an algorithm to find sufficient conditions to have a characterization involving a single forbidden subgraph. In order to evaluate the interestingness of a characterization, we have defined two criteria measuring its tightness and simplicity. We have then extended this algorithm to find sufficient conditions involving an arbitrary number of forbidden subgraphs. Using a similar approach, we have also developed algorithms to find necessary conditions to have a characterization involving a single or an arbitrary number of forbidden subgraphs. Lastly, we described an algorithm that finds actual FSCs.

Although these algorithms find FSCs, or conditions to have such characterizations, we have seen that this can be impossible in practice, due to the infinite number of graphs. However, we have shown that, by bounding the order of graphs under consideration, we could use our algorithms to find conjectures. Following this idea, we have used our algorithms to reproduce some important

results, as well as to find new ones, on the domination, independence and irredundance of graphs. In particular, our algorithms conjectured two new results for which we proved the correctness: 1) an n vertex graph G has the property that $ir(G) = \gamma(G)$ if it is "*Fork*"-free and "*Deer*"-free, and 2) G is $K_{(n-3)}$-free if $ir(G) < i(G)$.

While this work focused on forbidden *induced* subgraphs, an interesting extension could be to adapt the proposed methods to other forbidden structures, such as forbidden minors, or forbidden subdivisions.

References

[1] R. Allan and R. Laskar. On domination and independent domination numbers of a graph. *Discrete Mathematics*, 23:73–76, 1978.

[2] K. Appel, W. Haken, and J. Koch. Every planar map is four colorable. *Illinois: Journal of Mathematics*, 21:439–567, 1977.

[3] L. Beineke. Characterizations of derived graphs. *Journal of Combinatorial Theory*, 9:129–135, 1970.

[4] B. Bollobas and E. Cockayne. Graph-theoretic parameters concerning domination, independence and irredundance. *Journal of Graph Theory*, 3:241–249, 1979.

[5] G. Caporossi and P. Hansen. Variable neighborhood for extremal graphs 1: The system AutoGraphiX. *Discrete Mathematics*, 212:29–44, 2000.

[6] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006.

[7] G. Cornuéjols, X. Liu, and K. Vusković. A polynomial algorithm for recognizing perfect graphs. *Foundations of Computer Science, IEEE Annual Symposium on*, 0:20, 2003.

[8] S. Fajtlowicz. On conjectures of Graffiti. *Discrete Mathematics*, 72:113–118, 1988.

[9] O. Favaron. Stability, domination and irredundance in a graph. *Journal of Graph Theory*, 10:429–438, 1986.

[10] S. Földes and P. L. Hammer. Split graphs. In *Proc. of the Eighth Southeastern Conference on Combinatorics, Graph Theory and Computing*, pages 311–315. Congressus Numerantium, 1977.

[11] S. Fortin. The graph isomorphism problem. Technical Report 96-20, University of Alberta, Edomonton, Alberta, Canada, 1996.

[12] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[13] L. A. Goldberg. Polynomial space polynomial delay algorithms for listing families of graphs. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 218–225. ACM, 1993.

[14] D. L. Greenwell, R. Hemminger, and J. Klerlein. Forbidden subgraphs. In *Proc. of the Fourth Southeastern Conference on Combinatorics, Graph Theory and Computing*, pages 389–394. Congressus Numerantium, 1973.

[15] P. L. Hammer and B. Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981.

[16] P. Hansen, M. Aouchiche, G. Caporossi, H. Mélot, and D. Stevanovic. What forms do interesting conjectures have in graph theory? *Graphs and Discovery, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 69:231–252, 2005.

[17] T. W. Haynes, S. Hedetniemi, and P. Slater. *Fundamentals of Domination in Graphs*. Marcel Dekker, 1998.

[18] B. McKay. Practical graph isomorphism. *Congress Numeratium*, 30:45–87, 1981.

[19] B. McKay. Isomorph-free exhaustive generation. *Journal of Algorithms*, 26:306–324, 1998.

[20] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.

[21] J. Puech. Irredundance perfection and P_6-free graphs. *Journal of Graph Theory*, 29:239–255, 1998.

[22] N. Robertson, D. Sanders, P. Seymour, and R. Thomas. The four-colour theorem. *Journal of Combinatorial Theory*, 70(B 1):2–44, 1997.

[23] L. Volkmann and V. Zverovich. A proof of a conjecture on irredundance perfect graphs. *Journal of Graph Theory*, 41:292–306, 2002.