Studying graphs and their induced subgraphs with the computer : GraphsInGraphs

E. Camby[1], G. Caporossi[2]

[1] Département de Mathématique, Université Libre de Bruxelles

Bruxelles, Belgique

[2] GERAD & HEC Montréal

Montréal (Québec) Canada

February 28, 2016

Abstract

In the literature, graphs are often studied in terms of invariants, for instance the number of vertices or edges, the stability number, the chromatic number... However, it is common to study Graph Theory not only through invariants but also using subgraphs. Nevertheless, both concepts may be used together. Perfect graphs and cographs are two well-known examples of such line of study. In this paper, we present a computer software dedicated to the study of graphs and their induced subgraphs.

1 Introduction

Let $G = (V, E)$ be a simple graph, where V is the set of vertices and E the set of edges eventually joining them. The use of graphs for modeling purpose, where objects may be related to each other, is natural and of growing importance. The study of the properties of graphs and classes of graphs forms the graph theory.

One important concept in graph theory is that of invariant, a function that associates a numerical value to each graph regardless its labeling (regardless the way its vertices are numbered or drawn). Among invariants, the number of its vertices $n = |V|$, or its *order*, is likely the most natural one altogether with the number of its edges $m = |E|$. Invariants may, however, correspond to more complex properties such as the chromatic number, the minimum number of colors needed to assign each vertex a color such that both ends of any edge have different colors, or the stability number, the maximum number of vertices that are pairwise disjoint.

Invariant represents a key feature in graph theory and in applications such as chemistry, where an invariant is usually called a molecular descriptor [19]. The number of graph invariants is now very large and still growing. The study of graph theory by the mean of invariants is quite an active topic, and it is not surprising to notice that computers now play an important role in that study. Indeed, since the 1980's, the use of computers in graph theory keeps growing. The most important computer systems dedicated to the study of graph theory are likely the system *graph* [7, 8], Graffiti [9], AutoGraphiX [4, 5] and GraPHedron [15]. To the best of our knowledge, the system *graph* was the first system dedicated to graph theory. Graffiti was widely used as it suggested conjectures that attracted the attention of researchers and proposed important results whose the most famous is probably Fan Chung's Theorem [6]. Beside these two leader software, lots of others were developed more recently, we mention only two of them as they propose some innovative approach. *AutoGraphiX* uses optimization to find conjectured extremal graphs while *GraPHedron* is based upon the definition of a convex hull bounding the values of invariants.

However, graph theory is not only based upon the concept of invariants. Another key feature in graph theory is the concept of induced subgraph. Let $G = (V, E)$ be a graph. A graph $G' = (V', E')$ is called a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. It is then possible to build G' from G by removing some vertices and edges. Of course, removing vertices implies also the removal of their adjacent edges. If only the removal of vertices is considered, the subgraph G' is called *induced* subgraph. In other words, an *induced* subgraph $G' = (V', E')$ of G is a subgraph of G such as $E' = \{uv \in E | u, v \in V'\}$. Naturally, the graph G is called a *supergraph*, resp. an *induced supergraph*, of G' if G' is a subgraph, resp. an induced subgraph, of G. The best example of graph theory problem related to induced subgraphs is likely the *Ramsey* problem [18], which consists in, for every integer r and s, finding the minimum number $R(r, s)$ such that every graph on $R(r, s)$ vertices must have either an induced clique (set of vertices pairwise connected) on r vertices or an induced

stable set (set of vertices among which none are adjacent) on s vertices.

If subgraphs are often implicitly considered and used in computer systems (mainly because some invariants are defined by the mean of subgraphs), no computer software is designed specifically for the study of graphs and their subgraphs. In this paper, we present the system *GraphsInGraphs (GIG)*, which was specially designed to study graphs through their induced subgraphs.

In the second and in the third section, we present the main principles underlying the software and the associated data structure. Some direct applications and results are presented in Section 4, Section 5 and Section 6 while the last section concludes the paper.

From now on, in this paper, by abuse of appellation, we omit the word "induced" when we mention induced subgraph(s) or induced supergraph(s).

2 Description of the system

Identifying all subgraphs of order k in a graph G on n vertices may seem a tedious task. Indeed, in such a case, there are $\binom{n}{k}$ subgraphs to consider. The number of subgraphs of a given graph on n vertices is bounded by $\sum_{k=0}^{n}\binom{n}{k} = 2^n$. Searching for all subgraphs of all graphs of order n then seems unrealistic. However, this way to proceed yields a lot of redundancy. In this section, we explain how our computer system avoids this redundancy.

Let us start by recalling the number of non isomorphic (and non-necessarily connected) graphs according to their number of vertices as shown in Table 1. It is instructing to notice that the number of graphs grows very quickly, following an exponential behavior [10].

To avoid several times searching subgraphs of the same graph, we proceed by an implicit induction. Notice that, from a graph G, removing a vertex at a time will provide a family of n subgraphs on $n-1$ vertices. If all subgraphs from this family are already known, the task is complete.

The remaining critical point is the identification of those graphs. A canonical graph representation is used to quickly identify the subgraphs on $n-1$ vertices corresponding to any given graph on n vertices. This representation is based on the

n	# graphs
1	1
2	2
3	4
4	11
5	34
6	156
7	1 044
8	12 346
9	274 668
10	12 005 168

Table 1: Number of non isomorphic graphs with n vertices

software Nauty developed by Brendan Mckay [12]. Once all the graphs on n vertices are associated to their subgraphs on $n-1$ vertices, it is possible to go a step further and proceed to the search for graphs on $n+1$ vertices.

The all-subgraph-enumeration algorithm may be described as a recurrence relation as follows :

1. **Initialization**

   From a practical point of view, we start by the list of graphs on 2 vertices as being the first non obvious case (there are 2 such graphs). Then, each graph is represented in a canonical way.

2. **First step**

   Consider the list of all graphs on 3 vertices and represent them in a canonical way. Each of the 4 graphs on 3 vertices will be associated to 3 subgraphs on 2 vertices (by removing one vertex from the original graph). Again, a canonical

representation of the subgraphs allows a fast identification in the list of 2-vertex graphs and we establish a link between 3-vertex graphs and 2-vertex subgraphs.

3. **Main step**

   Suppose that, for $k = 3$ to $n - 1$, all the subgraphs on $k - 1$ vertices of each graph on k vertices are linked. We enumerate all graphs on n vertices and represent them in a canonical way. For each of these graphs, we build a list of all n subgraphs on $n - 1$ vertices, by removing one vertex from the original graph. The canonical representation of these subgraphs allows a fast identification from the existing dataset and we link all n-vertex graphs with all $(n - 1)$-vertex subgraphs. After this step, all subgraphs of $k - 1$ vertices associated to each graph on k vertices are known for $k = 3$ to n.

Repeating the main step for $n = 5$ to the desired value n_0 build a data structure that is then used for finding all subgraphs of order $k \geq 2$ of any graphs on $n \leq n_0$ vertices, and conversely for finding all supergraphs of order $n \leq n_0$ of any graphs on $k \geq 2$ vertices.

3 The network

The data structure used in the system could be described as a meta-graph of graphs. In order to avoid confusion, in the remaining of the paper, we will refer to this meta-graph as the *network*. Each considered graph is represented by a vertex in the network, and an edge between two vertices of that network indicates that one corresponding graph is a direct subgraph of the other, where a graph G' is a *direct subgraph* of G if G' may be obtained from G by removing only one vertex, i.e. G' is a $(n - 1)$-vertex subgraph of the n-vertex graph G. Similarly, G is a *direct supergraph* of G' if G' is a direct subgraph of G. Notice that in the network, we do not have all relations between graphs and subgraphs. Indeed, an edge links only a graph and one of its direct subgraphs.

Still keeping the notation simple, we note $sub(G)$ the set of edges between G and its direct subgraphs in the network. Similarly, $sup(G)$ the set of edges between G and its direct supergraphs. By a slight abuse of notation, when the context is clear, $sub(G)$ may also represent the set of direct subgraphs of G while $sup(G)$ may represent the set of its direct supergraphs.

As it may happen that a direct subgraph appears more than once in a given graph : each edge $e = GG'$ of the network has a weight w_e that represents the multiplicity of the corresponding direct subgraph G' of G, i.e. the number of the distinct vertices that may be removed from G to obtain G'. Then we always have

$$\sum_{e \in sub(G)} w_e = n,$$

if the graph G has n vertices. The network for $n_0 = 4$ is represented in Figure 1. For instance, sub() is { ,  } and sup( ) is the set of graphs called B, C, D, F, G and H.

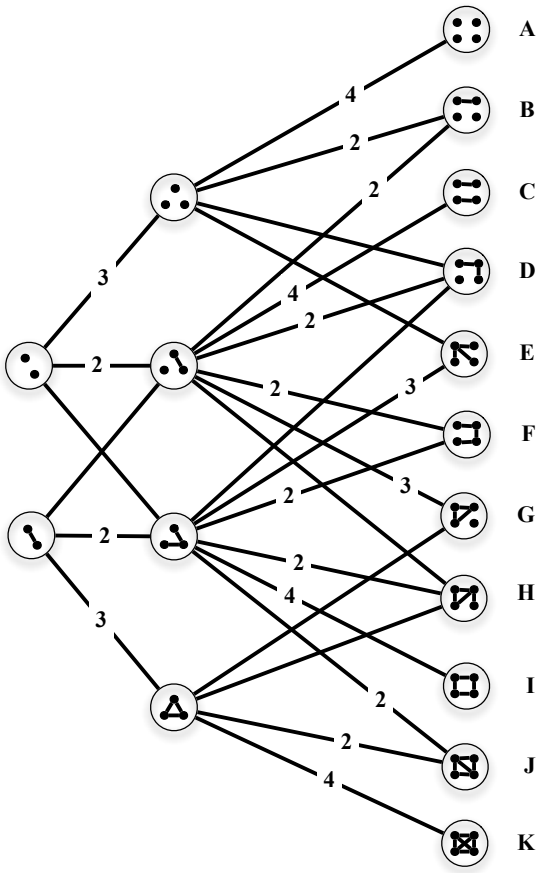

Figure 1: The network for graphs on 2 to 4 vertices, where the label displayed on each edge is its weight (unit weights are omitted).

The described network was built for all graphs on $2 \leq n \leq 10$ vertices, i.e. $n_0 = 10$. The graph generator *geng* [11] was used to generate these graphs. We developed a specific C++ program to build a list of the n subgraphs on $n-1$ vertices, for each n-vertex graph G. The so obtained direct subgraphs of G as well as G were then rewritten in a canonical way using the routine *labelg* from the nauty package [12]. Based on this data, the network builder writes a file describing the complete network. In user mode, the network is loaded from the disk and the analysis may be achieved. As expected, the construction phase is more time consuming than the exploitation, but in both cases, few minutes are enough.

4 Finding subgraphs and supergraphs

Let $\mathscr{F}$ denote a family of graphs to be used as a starting point. Depending of the search, $\mathscr{F}$ may represent the family of graphs in which subgraphs or supergraphs are studied.

Algorithm 1 identifies all subgraphs belonging to at least one graph from the family $\mathscr{F}$. Our algorithm is based on Breadth-first search [16].

```
   input  : A family of graphs 𝓕
   output: The family 𝓢 of all subgraphs of graphs from 𝓕
 1 𝓕_c ← 𝓕;
 2 𝓢 ← ∅;
 3 while 𝓕_c ≠ ∅ do
 4 |   𝓕_t ← ∅;
 5 |   foreach G ∈ 𝓕_c do
 6 |   |   foreach G' ∈ sub(G) do
 7 |   |   |   𝓢 ← 𝓢 ∪ {G'};
 8 |   |   |   𝓕_t ← 𝓕_t ∪ {G'};
 9 |   |   end
10 |   end
11 |   𝓕_c ← 𝓕_t;
12 end
13 return 𝓢;
```

Algorithm 1: Find all subgraphs.

Remark : If only subgraphs on a given order k are to be considered, lines 7 and 8 in Algorithm 1 may be replaced by the following instructions :

if G' has k vertices **then**
  $\mathscr{S} \leftarrow \mathscr{S} \cup \{G'\}$
else
  $\mathscr{F}_t \leftarrow \mathscr{F}_t \cup \{G'\}$

With our network, we can solve various questions on subgraphs. For instance, we can determine the common subgraphs of a family $\mathscr{F}$ of graphs by computing, separately, with Algorithm 1, families of all subgraphs for each graph of $\mathscr{F}$ and by intersecting these families. It is easy to recognize the union of subgraphs of graphs, as Algorithm 1 does, regardless the number of graphs in the initial family, but the running-time of the intersection depends on the number of graphs in the initial family. In the same spirit, we could establish diverse subgraphs : (common) subgraphs of a family of graphs which are not subgraphs of every (or any) graph of another family.

The case of finding supergraphs containing (or not) graphs from $\mathscr{F}$ is achieved in exactly the same way as finding subgraphs of graphs in $\mathscr{F}$. The only difference is that $sup(G)$ should be used instead of $sub(G)$ (for instance, line 6 of Algorithm 1). By mixing both notions of subgraph and supergraph, we can also search graphs belonging (or not) to the union (or the intersection) of sub/supgraphs from a family of graphs. These capabilities make the system a useful tool for studying small graphs according to forbidden and/or required sub/supgraphs. As an illustration, our system can find all subgraphs of any graph from a family $\mathscr{F}$, of order at most 10, which do not contain a triangle (i.e. a clique on 3 vertices).

5 Counting graphs

Our system allows to count small graphs under diverse constraints. We present here two of these counts.

5.1 Given 2 graphs, counting subgraphs

In this section, we introduce how to count the number $N(G', G)$ of times that a given k-vertex subgraph G' appears in a fixed n-vertex graph $G = (V, E)$, i.e. the number of subsets V' of V such that the subgraph induced by V' is isomorphic to G'. Notice that it is possible that multiple paths, i.e. ways of removing vertices, in the

network lead to the very same subgraph. Therefore, it is required to know how many paths lead to a subgraph constructed by the same set of vertices, i.e. the number of sequences of removed vertices, i.e. $(n-k)!$ ways.

Let $\mathscr{P}(G', G)$ be the set of shortest paths in the network between G' and G. Consider then $E(P)$ the set of edges in a shortest path P from G' to G. The multiplicity of this path P is

$$w_P = \Pi_{e \in E(P)} w_e,$$

where w_e denotes the weight of the edge e, because each edge of the network may be associated with more than a direct subgraph. Thus, the number of ways of successive removing vertices to obtain G' from G is given by

$$w_{G',G} = \sum_{P \in \mathscr{P}(G',G)} w_P.$$

Property 1 *The number $N(G', G)$ of times that the k-vertex subgraph G' appears in the n-vertex graph G' is :*

$$N(G', G) = \frac{w_{G',G}}{(n-k)!}$$

where $(n-k)!$ denotes the number of permutations of $n-k$ elements.

5.2 Counting graphs with given subgraphs

The first natural example of the use of the system is to count the number of graphs on n vertices according to the presence or not of some subgraphs (or a combination of those). For instance, the number of graphs on $n = 10$ vertices with either $\mathbf{X}$ (rows) or $\mathbf{Y}$ (columns) as a subgraph is given in Table 2. Graphs A to K are depicted in Figure 1.

It is not surprising to notice that the graph F, the path on 4 vertices, appears in a larger number of graphs since F is its own complement and its density equals $1/2$. By definition of the logical "OR", Table 2 is symmetric according to the main diagonal (AA to KK), but observe that this table also does according to the other diagonal (AK to KA), since the graph A is the complement to K, B to J, and so on.

X/Y	A	B	C	D	E	F	G	H	I	J	K
A	9110536	9094115	8948489	9110304	9101193	9106722	9078244	9097916	8928280	8996995	6319610
B	9094115	11890828	11696136	11890526	11870792	11886610	11858489	11878249	11689758	11776536	8996995
C	8948489	11696136	11803486	11791936	11770631	11800704	11777310	11791229	11612608	11689758	8928280
D	9110304	11890526	11791936	11992500	11970414	11988312	11958474	11979914	11791229	11878249	9097916
E	9101193	11870792	11770631	11970414	11970874	11966865	11936734	11958474	11777310	11858489	9078244
F	9106722	11886610	11800704	11988312	11966865	12000544	11966865	11988312	11800704	11886610	9106722
G	9078244	11858489	11777310	11958474	11936734	11966865	11970874	11970414	11770631	11870792	9101193
H	9097916	11878249	11791229	11979914	11958474	11988312	11970414	11992500	11791936	11890526	9110304
I	8928280	11689758	11612608	11791229	11777310	11800704	11770631	11791936	11803486	11696136	8948489
J	8996995	11776536	11689758	11878249	11858489	11886610	11870792	11890526	11696136	11890828	9094115
K	6319610	8996995	8928280	9097916	9078244	9106722	9101193	9110304	8948489	9094115	9110536

Table 2: Number of 10-vertex graphs containing either subgraph **X** or subgraph **Y**, both on 4 vertices, where labeled graphs **A**, ..., **K** are drawn in Figure 1.

6 Restricting the network to some families of graphs

The question may arise to know if it is possible to adapt the network in order to speed up the search when restricting to some families of graphs, particularly, if the family of graphs is closed by subgraphs. For instance, such families are bipartite graphs, forests or graphs with bounded maximum degree. In that case, not only previous operations are still available for this subnetwork but also it is possible to expect the system to handle larger graphs.

7 Concluding remarks and open questions

We present, in this paper, the fundaments of the software GraphsInGraphs, with various applications. To date, the software is still a prototype that will be implemented and made available as soon as possible. GraphsInGraphs will be the first computer system allowing survey of graphs depending on their induced subgraphs.

The research avenues in Graph Theory are manifold. Indeed, one of the unsolved problems in mathematics concerns the Reconstruction Conjecture [14, 17, 20], which states that any two graphs (on at least 3 vertices) with the same family of direct subgraphs and the same associated multiplicities are isomorphic. McKay [13] verified the Reconstruction Conjecture for all graphs with at most 11 vertices. However, GraphsInGraphs should point out some properties and observations that could be helpful in the study of this conjecture.

From another point of view, investigating graphs and their induced subgraphs

are intimately related to graph invariants. A famous example is perfect graph [1]. These graphs are defined by the equality of the value of two invariants for all induced subgraphs. Similarly, other research questions may also be handled by the system GraphsInGraphs. In addition, we should implement the comparison between the values of distinct invariant for all induced subgraphs, or the comparison between the values of an invariant for a graph and for its induced subgraphs, as it is the case for PoC-critical graphs [2, 3].

In brief, GraphsInGraphs is an innovative system for computer aided Graph Theory based on the concept of graphs and induced subgraphs.

Acknowledgements

This work was supported by NSERC (Canada), by Fédération Wallonie-Bruxelles (Belgium) and by Université Libre de Bruxelles (Belgium).

References

[1] C. Berge, Perfect graphs, *Six Papers on Graph Theory* (1963), pp. 1–21.

[2] E. Camby, Connecting hitting sets and hitting paths in graphs, *PhD thesis in ULB* (2015).

[3] E. Camby, J. Cardinal, S. Fiorini, O. Schaudt, The price of connectivity for vertex cover, *Discrete Mathematics & Theoretical Computer Science 16* (2014), pp. 207–224.

[4] G. Caporossi, Variable neighborhood search for extremal vertices: The AutoGraphiX-III system, *Les cahiers du GERAD G-2015-09* (2015).

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

[6] F. Chung, The Average Distance and the Independence Number, *Journal of Graph Theory 12* (1988), pp. 229–235.

[7] D. Cvetković, I. Gutman, The computer system GRAPH: A useful tool in chemical graph theory, *Journal of computational chemistry 7* (1986), pp. 640–644.

[8] D.M. Cvetković, L.L. Kraus, S.K. Simic, 733. DISCUSSING GRAPH THEORY WITH A COMPUTER I. IMPLEMENTATION OF GRAPH THEORETIC ALGORITHMS, *Univ. Beograd Publ. Elektrotehn Fak.* (1981), pp. 100–104.

[9] S. Fajtlowicz, On conjectures of Graffiti,*Discrete mathematics 72* (1988), pp. 113–118.

[10] F. Harary, A.J. Schwenk, The number of caterpillars, *Discrete Mathematics 4* (1973), pp. 359–365.

[11] B.D. McKay, Geng, *A program for generating graphs available at http://cs. anu. edu. au/˜ bdm/nauty* (1984).

[12] B.D. McKay, Nauty user's guide (version 2.4), *Computer Science Dept., Australian National University* (2007), pp. 225–239.

[13] B.D. McKay, Small graphs are reconstructible, *Australasian Journal of Combinatorics 15* (1997), pp. 123–126.

[14] P.J. Kelly and others, A congruence theorem for trees, *Pacific J. Math 7* (1957), pp. 961–968.

[15] H. Mélot, Facet defining inequalities among graph invariants: the system GraPHedron, *Discrete Applied Mathematics 156* (2008), pp. 1875–1891.

[16] E.F. Moore, The shortest path through a maze, *Bell Telephone System.* (1959).

[17] P.V. O'Neil, Ulam's conjecture and graph reconstructions, *The American Mathematical Monthly 77* (1970), pp. 35–43.

[18] F. P. Ramsey, On a problem of formal logic, *Proceedings of the London Mathematical Society 30* (1930), pp 264?286.

[19] R. Todeschini, V. Consonni, Molecular Descriptors for Chemoinformatics, *John Wiley & Sons 41* (2009).

[20] S.M. Ulam, A collection of mathematical problems, *Interscience Publishers 8* (1960).