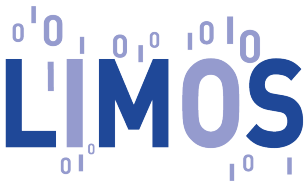


Rapport de stage

Potential Maximal Cliques enumeration

Écrit par Nicolas Schivre

Février - Juin 2025

Maîtres de stage :
Vincent LIMOUZY
Pierre BERGÉ

Référent de l'université :
Frédéric DABROWSKI

M2 informatique Applications Réparties, Intelligence Artificielle et Sécurité

Contents

Introduction

In computer science, there are plenty of fields to explore. Between those fields, there is one that catches my interest : graph theory. First, its eponymous famous tool is a well studied combinatorial structure. Graphs are mostly represented by nodes called vertices linked by lines called edges. They can be seen as a multitudes of objects linked between them depending of some binary relation. Regardless of their simplicity, graphs are used in a wide scope of real-life domains as a way of modeling situations. Indeed, it has applications in chemistry for analysing the molecules structures, in biology for phylogenetic networks, in operational research for scheduling... The typical goal of this kind of modelling is to represent a situation which presents a difficult problem in order to find an efficient way of solving it.

For the purpose of illustrating this point, please see the Figure 1 that represents one of the most famous application of graph for a real life problem. Indeed, this image represents the famous city of Königsberg which was the inspiration for one of the first problem introduced in the graph theory field. It is due to Leonhard Euler who published in 1736 one of the first article which mentions graph theory [15]. The question Euler asked was the following : How can someone travel across all the seven bridges of Königsberg without crossing any bridge twice ? Indeed this problem can be seen as a graph problem. Let us replace all the different lands separated by the river by a node and two lands are linked together if there is a bridge between them. There can be multiple edges between a pair of nodes if there is multiple bridges between them. Nowadays, this kind of graph is known to be a multigraph and the path searched by Euler now have an eponymous name : the *eulerian path*. With the actual knowledge about graph theory, we can easily say that there is no such path for the Königsberg city graph.

As an example of the application of graphs in other fields of research, Figure 2 is an example of graph modelling of a biological situation from [29] where the authors were working on mapping the protein-protein interactions with human body proteins and more precisely analysing the interaction of some disease-associated proteins. In this graph, the green nodes are the disease-associated proteins and the yellow nodes are the proteins with no disease association. There is a red or blue edge between two proteins in order to represent different kind of interactions between them.

The great popularity of this data structure probably comes from its incredible versatility. Indeed, as shown above, graphs can be used in almost all fields. Even

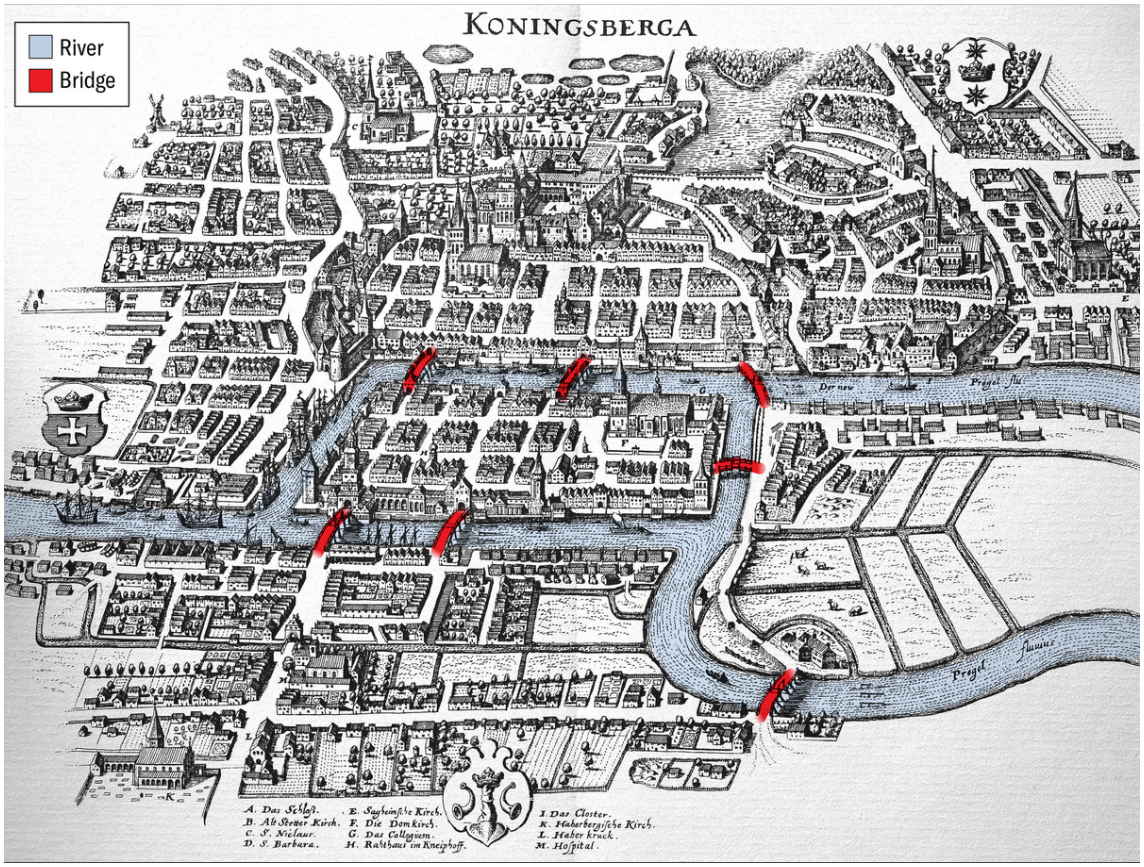


Figure 1: Königsberg city in the XVIII's century and it's famous seven bridges crossing Pregel river

if the classical graph does not fit in a problem, there are various classes to use depending on the situation. Actually, there is an online database called **Graph Classes** (ISGCI [14]) which classify more than 1600 classes of graphs.

In graph theory, there is plenty of ways to use graphs. Some analyse their structure to better understand their structural properties while others use those structural results to solve problems. This latter is well studied because of the existence of many difficult problems. The most popular kind of problems graphs theory researcher are trying to solve are *optimization problems*. Here are few examples of a typical optimization problem : What is the shortest path between two points on a map ? What is the minimum path to pass by all letterboxes of a city without passing by the same letterbox twice ? What is the maximum or minimum set of vertices that hold a certain property ? Those problems can be easily figured out in some specific situation, but in the general case most of them are hard.

The goal of this field is to conceive clever and efficient algorithms as a way to find a solution to these problems. However, as said in the previous paragraph, most of those useful problems are hard. In fact, for some, there is no algorithm to solve

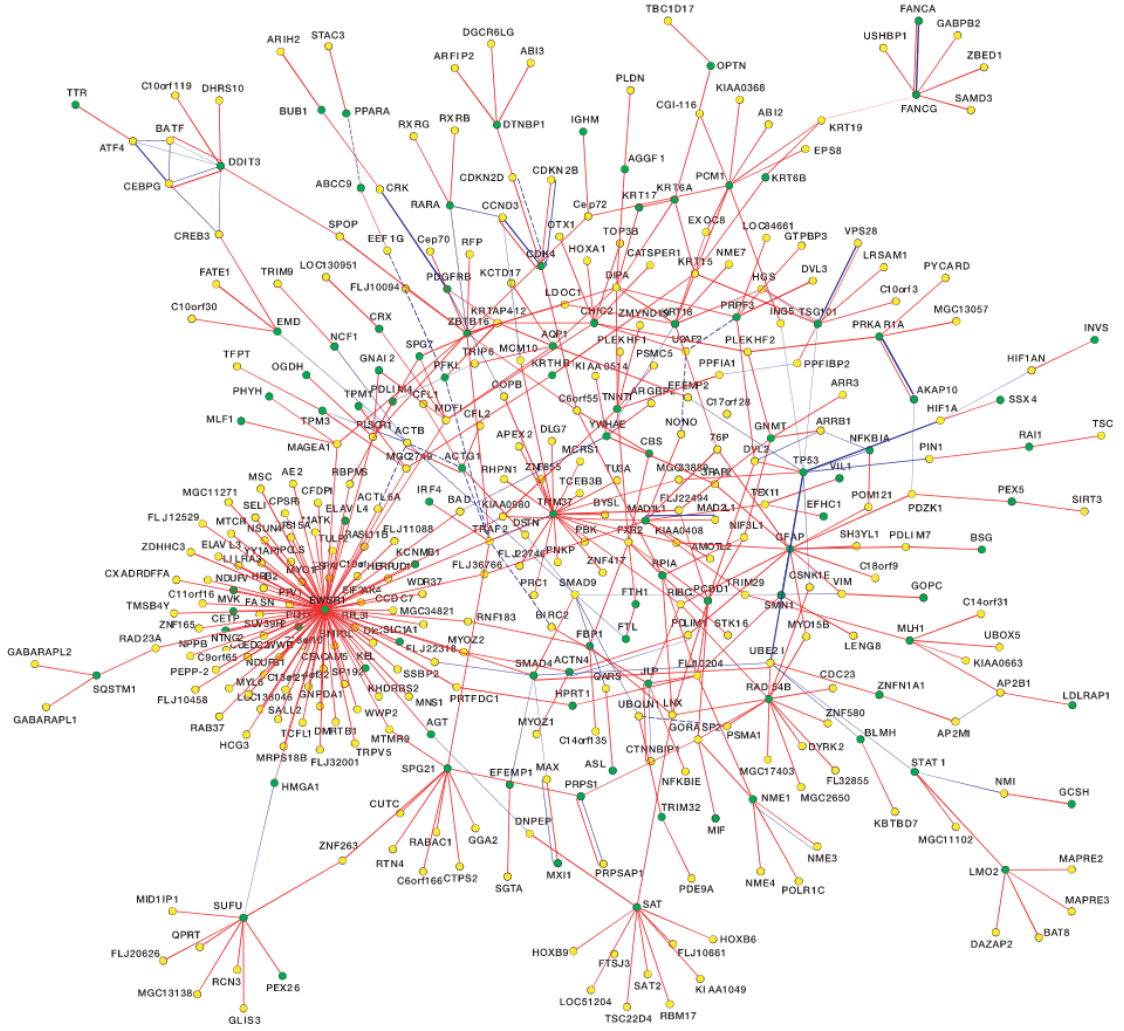


Figure 2: Interaction network of disease-associated proteins

them efficiently, and, in fact, it is believed that those problem cannot be solved *efficiently*. Some complexity classes were made to classify the problems depending on their hardness. The two main classes are the "easy" problems in **P** and the "hard" problems in **NP-complete**. Studying these classes is one of the most active field of research as a famous question involving these two remains opens since the last century : Is $\mathbf{P} = \mathbf{NP}$? Answering positively to this question will have a major outcome on the society as for example a lot a cryptography protocol are made with problem which come from NP. If these two classes were identical, then it will ensure that a efficient algorithm solving these problems exists. In reality, most of researcher are assuming that $P \neq NP$ as a result of the hardness of solving NP problems.

Sometimes, as finding an optimal solution is not enough for a given application, a field of algorithm conception consists in working on enumerating all the good

solutions. Actually, they tackle the **enumeration** version of the problem. The goal of this kind of problem is to produce the solutions exactly once and with the best total time and space complexity. In this field of study, there is two trends : the **input sensitive** and the **output sensitive**. The first will give an algorithm depending on the size of the input only while the other give an algorithm with complexity measured with the input and the output sizes. There are multiple pros and cons to both type of algorithm based on the need.

In some problems, there is a number of solutions which can be exponentially large in the input size. For example, in melon graphs, there is an exponential number ($3^{n/3}$) of minimal separators [17]. In this case, we cannot expect a complexity less than exponential. Therefore, with the classical algorithm evaluation, all enumeration algorithms will not be considered as efficient. However, if the number of solutions for some specific instances is "small" (i.e. polynomial in the graph size), a **polynomial delay** algorithm will compute all the solutions in polynomial total time on these instances. In order to analyse enumeration algorithms complexity, another kind of algorithm complexity need to exist. For the input sensitive algorithms, the complexity is of the form a^n where n is the size of the output and the goal is to lower a as minimum as possible. For the output sensitive approach, it is completely different as the complexity depends on the output size. Indeed, in output sensitive algorithms, the goal is to lower the delay between the finding of two different solutions. The complexity analysis is focused on the complexity of this delay.

In this master's thesis in graphs, I will focus myself onto the enumeration of particular objects called potential maximal cliques. A definition followed by the state of the art of the different enumeration techniques and the potential maximal clique object will be made at first. Secondly, we will give an overview of the different results of this internship. Finally, the conclusion of this document will present the remaining work to do and the opportunities of pursuing in studying this topic.

Before going into the main content of this document, I thanks the *Laboratoire d'Informatique de Modélisation et d'Optimisation des Systèmes* (LIMOS) and more precisely the line of research *Modèles et Algorithmes de l'Aide à la Décision* (MAAD) who welcome myself for this internship.

Chapter 1

Introduction to graphs

This chapter will give a short introduction concerning the graph object. From basic definitions and notations to some necessary graph classes, this chapter will give all the necessary information concerning graphs in order to help the understanding of the main content of this manuscript. Moreover, the end of this chapter will have a section defining with a little more details the separators which will be a key tool to the potential maximal cliques understanding.

1.1 Definition, properties and classes

1.1.1 Definition and properties

First let us give the definition of a graph itself.

Definition 1 (Graph (undirected))**.** *An undirected graph is a binary relation of two elements $G = (V, E)$ with V the set of vertices (or nodes) and E the set of edges. An edge e is a bidirectional relation between two nodes u and v from V : it is noted (u, v) or in short uv.*

Remark 1. *In the case of undirected graphs, uv and vu define the same notion.*

A vertex of a graph often describes an object and an edge the existence of a link between two objects. Most of the time, a node is represented as a circle and a edge as a straight line between two nodes. The former definition is a definition of undirected graphs. In this case, the edge represents a symmetric relation. On the contrary, there exists directed graphs where edges represent an asymmetric relation. Unless written explicitly all graphs will be considered as undirected ones.

Knowing the definition of a graph, let us take a good look at various properties of graphs.

Neighbourhood

The notion of neighbourhood is related to the edges. The neighbourhood of a vertex $v \in V$ denoted $N(v)$ is the set of vertices that are at an endpoint of an edge where the other endpoint is v. This former definition is the *open neighbourhood* of v. More formally :

Definition 2 (Open Neighbourhood)**.** *The open neighbourhood of a vertex $v \in V$ is $N(v) = \{u \in V | uv \in E\}$.*

Given an open neighbourhood, one can define its *closed neighbourhood* :

Definition 3 (Closed Neighbourhood)**.** *The close neighbourhood of v is $N[v] = N(v) \cup \{v\}$.*

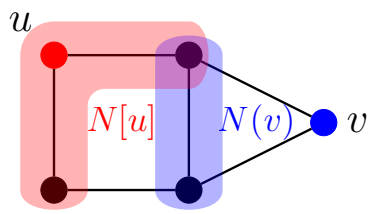


Figure 1.1: Difference between open and closed neighbourhood

In Figure 1.1, we can see in highlight the closed neighbourhood $N[u]$ of u and the open neighbourhood $N(v)$ of v.

When dealing with neighbourhood, some might want to compute their sizes. It is called the *degree* of a vertex. More formally :

Definition 4 (Degree)**.** *The degree of $v \in V$ is $d(v) = |N(v)|$.*

Paths and cycles

Paths are a common tool of graphs that are used in many problems. Here is a definition : let $P \subseteq V$ be a path from a to b of G. P is a succession of vertices such that a new vertex is chosen in the neighbourhood of the previous one, such that the first vertex is a, the last is b, and there is no repetition of any vertex in between. More formally :

Definition 5 (Path)**.** *We say $P = (v_1, v_2, ..., v_k)$ is a k-length path if $v_i v_{i+1} \in E(G)$, $\forall i \in [1, k-1]$ and $v_i \neq v_j$, $\forall i, j$, $i \neq j$.*

Without loss of generality let us define cycles :

Definition 6 (Cycle)**.** *A cycle is a path from a vertex a to this same vertex a.*

Connectivity

With the notion of neighbourhood and path, we can define the notion of connectivity of a graph :

Definition 7 (Connectivity)**.** *A graph G is connected if for any pair of vertices $u, v \in V$, there is a path between them.*

In a non connected graph, we define the connected components :

Definition 8 (Connected components)**.** *Given a non connected graph G, the connected components of G are all the distinct maximal induced subgraphs that are connected.*

In fact, we usually say that a connected graph has a unique connected component.

Clique

A clique is a set of vertices such that there is an edge between any pair of vertices in this set, hence, we define :

Definition 9 (Clique)**.** *$S \subseteq V$ is a clique if $\forall u, v \in S : u \neq v$, $uv \in E$.*

Lemma 1 (Maximal Clique)**.** *A clique S is said maximal if for every vertex $v \in V \setminus S$, $S \cup \{v\}$ is not a clique.*

From the previous lemma we can assume that a clique cannot be strictly included in one another.

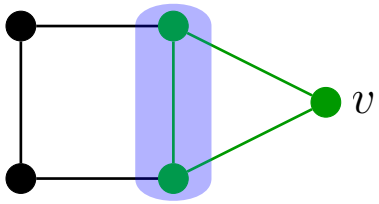


Figure 1.2: Difference between a clique and a maximal clique

In Figure 1.2, $N(v)$ is a clique but it is not a maximal clique. Adding v to $N(v)$ creates a maximal clique in this graph.

Subgraph / Supergraph

A *subgraph* H of a graph G is a graph that is defined on a subset of nodes and edges of G. Conversely, G is a *supergraph* of H because all the nodes and edges of H are included in G. H is said *induced* if H is the graph created from a set of nodes

$X \subseteq V$ and all the edges $\{u, v \in X | uv \in E\}$. Hence, the induced subgraph of a clique is called complete (all the nodes pairs are linked by an edge).

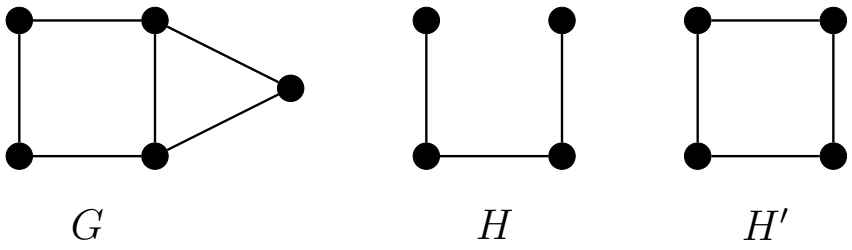


Figure 1.3: Difference between a subgraph and an induced subgraph

In Figure 1.3, G is the supergraph of H and H' : they are both subgraphs of G. Only H' in an induced subgraph of G.

Directed graph

Definition 10 (Graph (directed))**.** *A directed graph is a tuple of two elements $G = (V, A)$ with V the set of vertices and A the set of arcs. An arc a is a asymmetric relation between two nodes u and v from V. The arc a is noted uv denotes an arc starting from u and ending in v. Conversely vu denotes an arc starting from v and ending in u.*

This document will not go into detail about the directed graphs properties as almost all used graphs will be undirected. However the only knowledge that will be necessary to understand about directed graphs is that the notion of neighbour is now asymmetric. Indeed for an arc uv, v is in the neighbourhood of u while the opposite is not true. From this follow that, taking two vertices a and b, the existence of a path from a to b does not ensure the existence of a path from b to a.

1.1.2 Graph classes

Graph classification is a powerful tool to understand graph theory. Undoubtedly, knowing various graph classes properties can be a great help to understand and find an efficient solution to some problem. In some cases, problems are specifically related to a precise class of graphs. As stated in the introduction, there are over 1600 graph classes that have been described in [14]. In this subsection, we will first describe a class of graph that is strongly connected to the potential maximal cliques : the chordal graphs. Secondly, we will describe a particular class of graphs that hold some of the results around potential maximal cliques application : the P_5-free graphs.

Chordal graphs

Chordal graphs are the most important class of graphs of this manuscript as the problem studied is based on a property of this class.

Definition 11 (Chordal graph)**.** *A graph G is chordal if every cycle of length greater than three has an edge joining two non-consecutive vertices of the cycle. We say that the cycle has a chord.*

Remark 2. *A cycle of length four with no chord is called a C_4.*

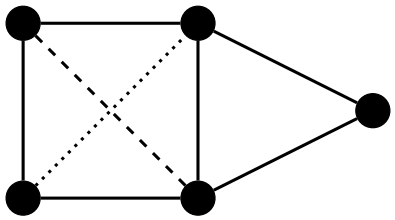


Figure 1.4: Chordal graphs

The graph of Figure 1.4 is not chordal because of the presence of a C_4. Adding either the dashed or the dotted edge make the graph chordal. We can notice that a graph can always be transformed to a chordal graph by adding edges in the chordless cycles. This operation is called the triangulation of a graph. Here a more formal definition :

Definition 12 (Triangulation [19])**.** *G' is a triangulation of $G = (V, E)$ if $G' = (V, E \cup X)$ where X is an edge set which makes G chordal.*

As a lot of graph properties, this one can either be maximised of minimised. In the case of triangulation, only the minimisation is important.

Definition 13 (Minimal triangulation [20])**.** *$G' = (V, E \cup X)$ is a minimal triangulation of $G = (V, E)$ if for any edge $e \in X$, $G' - e$ is not chordal.*

We can notice that these graph can be recognized in linear time $O(n+m)$ by an algorithm based on the *perfect elimination ordering* from Rose, Tarjan and Lueke in [28]. Moreover, the chordal graph classes admit a tree decomposition where each vertex represent a clique of the chordal graph. In order to create this tree, we will create the clique intersection tree. A clique intersection tree is a tree where all vertices are intervals and two intervals intersect if the corresponding vertices are linked by an edge. This construction verifies the Helly property.

Lemma 2 (Helly property [30])**.** *A geometric intersection model has the Helly property if for every clique C there is a single point p such that every vertex of C include this point p.*

(a) Chordal graph (b) Clique intersection tree (c) Clique tree

Figure 1.5: Decomposition of a chordal graph

Finally, given this clique intersection tree, we can find all the maximal cliques easily by searching for the p point from the Helly property definition, and then link the maximal cliques if they have a vertex in common. Please see Figure 1.5 to see the construction from the original chordal graph.

P_5-free graphs

A P_5-free graph is a graph that does not have a path on 5 vertices (P_5) as an induced subgraph. This class is a well studied class of graph since a lot of NP-Complete problem can be solved in polynomial time on it. For example the problem Maximum Independent Set which is defined below is NP-Complete in the general cases, but, Lokshtanov *et al.* presented a polynomial time algorithm solving this problem in [24]. Furthermore, their algorithm is particularly interesting because it features the use of potential maximal cliques to solve the problem.

Maximum Independent Set
Input: A graph $G = (V, E)$.
Output: A set $S \subseteq V$ such that $\forall u, v \in S \| u \neq v,\ u \notin N(v)$ of maximum cardinality.

1.2 Separators

As a final section of this chapter and as a useful property that will be frequently used in the following chapters, vertex separators will be define below.

Definition 14 (a, b-separator)**.** *Given a set of vertices $S \subset V$ and two vertices $a, b \in V$, S is an (a, b)-separator if a and b are in different connected components in $G - S$.*

Remark 3. *The S is said minimal if for any subset $S' \subset S$, S' is not an (a, b)-separator.*

Definition 15 (Separator)**.** *We said that S is a separator if S is an (a, b)-separator for some pair $a, b \in V$.*

Remark 4. *We note Δ_G the set of all minimal separators of G.*

Lemma 3. *[[8]] For any graph G with $a, b \in V$, and any (a, b)-separator $S \subset V$ of G, S is minimal if and only if for every $v \in S$, there is a path from a to b which intersect S only in v.*

The connected components created by a minimal separator can hold a property named *full*.

Definition 16 (Full component)**.** *A connected component C is a full component to a set of vertices X if, $\forall x \in X$, $N(x) \cap C \neq \emptyset$.*

This notion of full component will be really helpful when we will describe the potential maximal cliques.

Lemma 4. *A minimal separator S admit at least two full components.*

Proof. A separator creates at least two connected components. Let S be a minimal a, b-separator. It follows that $G \setminus S$ has one component containing a denoted C_a and one containing b denoted C_b. By Lemma 3, it follows that both a and b have a path from themselves to any vertex of S. □

Definition 17 (Close separator [31])**.** *Let $A \subset V$ be a connected set (i.e such that $G[A]$ is connected) where $a \in A$, $b \notin A$. The close separator S of A is the unique minimal (a, b)-separator included in $N(A)$, called $S(A)$. The set $S(A)$ is obtained by computing the neighborhood of the connected component containing b in $G \setminus N(A)$ (see Figure 1.6).*

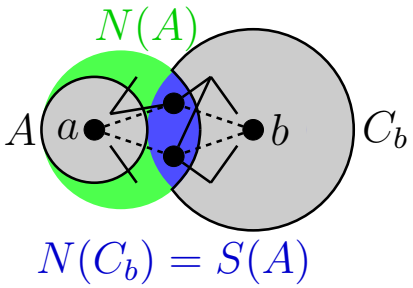


Figure 1.6: Schema of a close separator (in blue)

We can also describe a more restrained type of minimal separators : the inclusion wise minimal separators.

Definition 18. *Let S be a separator of G. S is a inclusion wise minimal separator if there is no subset $X \subsetneq S$ such that X is also a separator.*

This has already been proved by Brosse *et al.* [9] that enumerating all this kind of separators is not possible with an output polynomial algorithm unless $P = NP$.

Definition 19 (Parallel separators)**.** *Two separators S and T of a graph G are parallel, if S (resp. T) does not contain any vertices from two distinct components in $G \setminus T$ (resp. $G \setminus S$).*

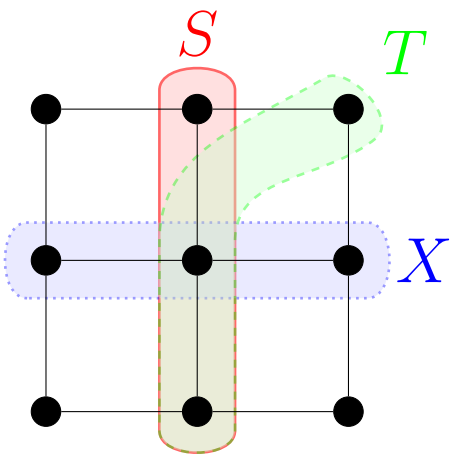


Figure 1.7: Parallel separators

In Figure 1.7, we can see that the separators S and T are parallel since T have vertices in only one connected component of $G \setminus S$. However, S and X (and also T and X) are not parallel.

Chapter 2

State of the art

2.1 Enumeration

In graph theory, usual tackled problems only search for a unique "best" solution considering some criteria. Those problems are called *optimization problems*. Unfortunately, many years of study in this domain have shown that a lot of optimization problems are hard to solve in polynomial time. Hence, most of them are NP-Complete. As those problems can be hard, researchers either solve a parameterized version (thus algorithm with a complexity based on the output and some other parameters) of these, solve the approximation version where they find an approximate solution (a solution which is at a calculable distance ratio to the optimal solution), solve it on a restrained classes of graphs or solve what is called the enumeration version of these problems. The enumeration version of a problem is different from the optimization version in the way that not only the "best" solutions need to be outputted. Unlike optimization problem, enumeration problem does not search for the minimum/maximum solutions explicitly. In fact, enumeration problems are often taken into account when the optimization version is hard to solve efficiently. In this kind of problem, the minimal/maximal solutions is sought. The difference between a minimum/maximum and a minimal/maximal solution is the fact that given a minimum/maximum solution, there is no other solution with a lower/greater size than this solution. Considering a solution, the solution is minimal/maximal if removing/adding any element of/to this solution make it no longer one. Finally, in the process of the enumeration, we does not want any duplicate solution. To achieve this, we need to find a way to remember the outputted solutions. We can easily see that in some problems with exponential number of solutions, the trivial remembering of all the already outputted solutions leads to an exponential space use. It follows that optimizing the space complexity of such algorithm can also be challenging.

Knowing the goal of enumeration, two approaches for solving these problems emerged. First, the *input sensitive* approach and secondly the *output sensitive* approach. The first one only considers the input size to bound the complexity. In

this case, as most of enumeration problems have an exponentially large number of solutions, then the complexity of input sensitive algorithm is naturally exponential in the input size. With this exponential complexity of the shape $O(a^n)$, it cannot be seen as efficient regarding of the usual algorithm complexity. Researchers working on input sensitive algorithms try to lower the a in the complexity in order to make the time complexity as lower as the number of solutions.

The second approach and the one that will be studied in the rest of this manuscript is an approach which consider both the input and the output size in the complexity analysis. Since there is an exponential number of solutions, an output sensitive algorithm goal is to lower either the total running time in function of the output or the time between each solution to be outputted. For the rest of this section, we will give an overview of this approach and some important result of output sensitive enumeration algorithms.

2.1.1 Time complexity

Analysing the complexity of an enumeration algorithm in the classical way (described by Jack Edmonds in the 1960's [1]) will often result in an exponential complexity lower-bounded by the number of solutions. In this case, how can the efficiency of an enumeration algorithm be evaluated ? This is a major motive to analyse the complexity with the input and output size. In the following, the input size will be regarded as n and the output size as N. In 1988 in [22], different complexity measures were introduced for output sensitive enumeration algorithms. They will be presented below.

Output polynomial

Also described as *Total Time* algorithm, *Output* algorithms are defined with a $f(n, N)$ complexity. Those algorithms have no guarantee on the time between two solutions to be outputted but it takes a maximum time bounded by $f(n, N)$. An Output algorithm complexity is bounded by the function f on n and N. Indeed the order of magnitude of f defines the algorithm complexity. For example, if f is a polynomial function, then the algorithm will be *Output polynomial* and if f is linear then it will be *Output linear*. Figure 2.1 schematises the behaviour of this kind of algorithm complexity in order to help its understanding.

These algorithms needs a $O((n + N)^c)$ time to produce all the solutions.

Incremental polynomial

Incremental algorithms are defined by a $\sum_{i=1}^{N} f(n, i)$ complexity, a sum of a function f on the input n and the number of previously outputted solutions i. Oppositely as the Output algorithms, the solutions are outputted with a $f(n, i)$ times between two

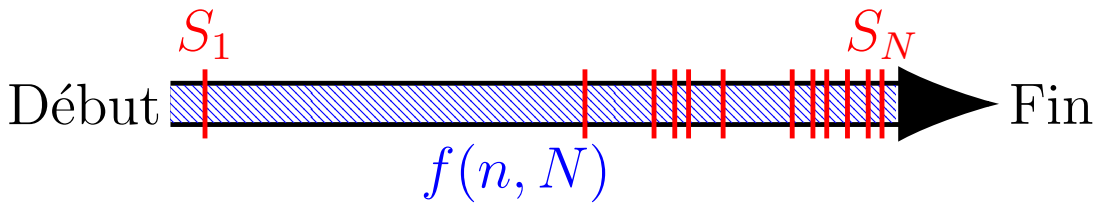


Figure 2.1: Example of the execution time of an Output algorithm and a potential solution distribution (in red) along the running time

solutions. In this way, the time between two solutions to be outputted may increase with each new solution. In the other hand, similarly as the Output complexity, Incremental are named based on the complexity of f. For instance, if f is polynomial, then the algorithm will be *Incremental polynomial*. Figure 2.2 schematises the behaviour of this kind of algorithms complexity in order to help its understanding.

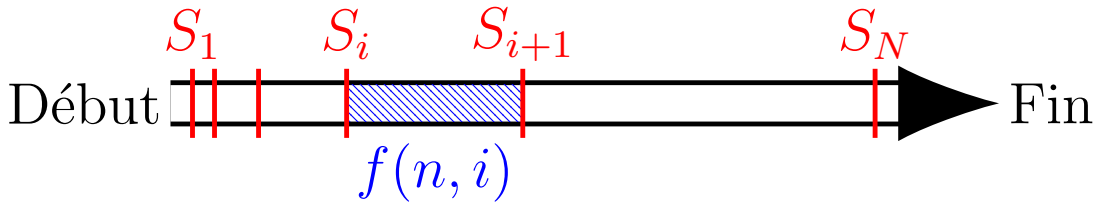


Figure 2.2: Example of the execution time of an Incremental algorithm and a potential solution distribution (in red) along the running time

Given L_i the list of solutions already found at the step i, these algorithms need a $O((n + |L_i|)^c)$ time to produce the $i + 1^{\text{th}}$ solution.

Polynomial delay

Lastly, *Delay* algorithm are algorithms with $f(n) \times N$ complexity. It follows that the solutions are outputted with an identical delay bounded by $f(n)$. Identically as the two previous complexity classes, Delay algorithms are defined by the complexity of f. Hence, if f is polynomial, then the algorithm will have a polynomial Delay. Figure 2.3 schematises the behaviour of this kind of algorithms complexity in order to help its understanding.

Constant delay

Lastly, we mention the *constant Delay* algorithm. Those are related to the polynomial Delay one except that constant Delay is the best hope for an Delay algorithm. Indeed, with this delay, only the modification between two solution can be used because any use of the graph entirely will imply a linear delay. In particular, a

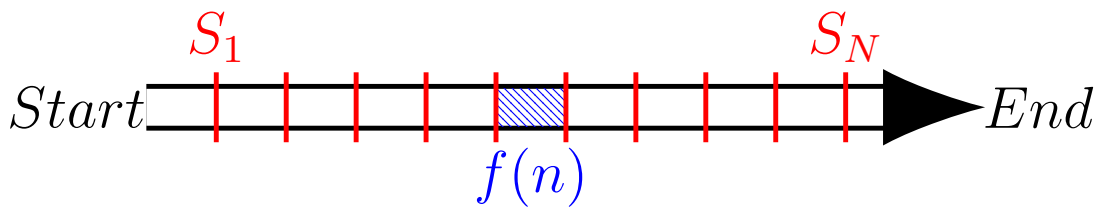


Figure 2.3: Example of the execution time of a Delay algorithm and a potential solution distribution (in red) along the running time

constant Delay algorithm starts with a preprocessing operation before navigating in constant delay between the solutions. To have more details and example about constant Delay algorithm, please refer to [25].

2.1.2 Space complexity

In the same way as classical algorithms, enumeration algorithms can be analysed regarding their space complexity. However, for enumeration problems, we might want to store the previous outputted solution in order to output them only once. This can, for example, be done by storing all the already outputted instances but this can be very space consuming as some problems will need a exponential number of solutions to be stored this way. Hence it will lead to a exponential-space algorithm. In order to be more efficient in space, some technique need to be settled as ordering the solution and does not output solution with a lower ordering than the actual solution.

2.1.3 Algorithmic methods

In order to design efficient enumeration algorithms with respect to the output sensitive enumeration time complexity, there are some methods in the literature. An non-exhaustive overview of these different methods will follow.

Flashlight Search

In 1975, one of the very famous enumeration technique was introduced by R. C. Read and R. E. Tarjan in [27]. This technique have various names in literature such as *Backtracking* [27], *Binary Partition* [23], or, the one that will be used in this paper, *Flashlight Search* [10, 11]. Considering an arbitrary ordering of the elements that could be in the solution $e_1, e_2, ..., e_n$, this method consist in creating node by node a binary tree rooted in $\emptyset$ and branched by e_i and $\overline{e_i}$ at depth $i-1$. Each e_i branch represent the solution including e_i and $\overline{e_i}$ excluding it. Without further modifications, we will end with the leaves of the tree representing all the different set of elements that stand as a potential solution. It means that some leaves may

not be a solution. In the Flashlight search, we want the leaves to represents all the different maximal solutions searched. In order to achieve this, we will stop extending a branch when no solution hides beneath. At each step, in order to prune the tree, the extension version of the problem need to be answered at each depth. Such problem template is defined as below :

Extension problem template
Input: A graph $G = (V, E)$, a subset $M \subset V$ of *mandatory* vertices, a subset $F \subset V$ of *forbidden* vertices such that $M \cap F = \emptyset$ and a property P to satisfy. **Question:** Does X, a set of vertices with $M \subseteq X$, $X \cap F = \emptyset$ and such that X satisfy P exist ?

In this definition P represents the property which has to be satisfied in order to be a solution. It can be various : being a minimal dominating set, a maximal independent set,... Given a node of the tree, if the answer of the extension problem considering the e_i elements in M and the $\overline{e_j}$ in F is *Yes*, then it is easy to see that there will be at least one solution below and it is worth continuing the exploration. It follows that if the answer is *No* then there is no solution below and we can cut the branch and backtrack to the parent node. In Figure 2.4 we can see an example of the exploration of the solutions using the Flashlight Search.

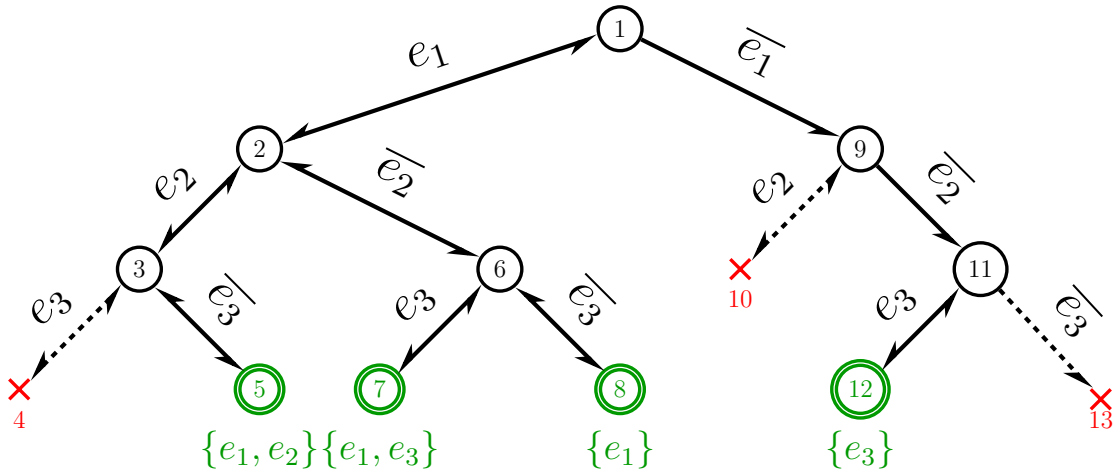


Figure 2.4: Example of an exploration of a binary tree with a Flashlight Search algorithm with a possible ordering on the nodes travelled

This technique is very effective. Indeed, if a polynomial time algorithm solving the extension version of the problem exists, then a polynomial delay, polynomial space algorithm comes out from Flashlight Search. As an example of such algorithm, the article [31] gives a clever algorithm to enumerate all minimal a, b-separators of a graph. A minimal a, b-separator of a graph is a minimal set of vertices that disconnects a and b in the graph. In this paper, the author achieves this by creating a polynomial delay, quadratic space algorithm using Flashlight Search.

Reverse Search

In 1996, *Reverse Search* framework was described by David Avis and Komei Fukuda in [2]. It has been motivated by *local search* which is a famous technique from optimisation field. Indeed this technique rely on navigating in the solutions space by improving a solution at each step until it became a local optimum solution. The Reverse Search principles are the followings : create a *rooted* spanning tree containing all the solutions and define an unique *parent* for each solution except for the root. The search will be done by exploring in a Depth First Search the spanning tree starting from the root. Please note that the tree can rather be a spanning forest depending on the situation. In this case, multiple parallel DFS on the roots are done.

With more detail, the tree structure is made with the help of three functions:

- A neighbouring function Neighbourhood which generates in polynomial time all the neighbouring solutions of a solution in order to create a directed supergraph of solutions.
- A parent function Parent which will help to define the spanning tree structure of the solutions graph.
- A children function Children which define the neighbours of a solution in the spanning tree. For a solution S, it is defined as follow : $\textsc{Children}(S) = \{S' \in \textsc{Neighbourhood}(S) |\ \textsc{Parent}(S') = S\}$.

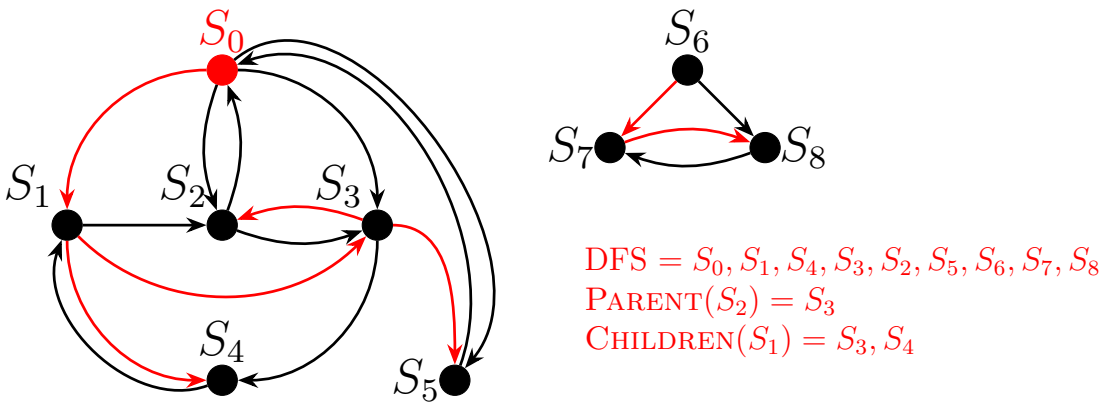


Figure 2.5: Example of a possible solutions supergraph and its spanning forest structure in red

The complexity of such algorithm relies on the complexity of the Children function and the maximum number of neighbours that this function creates in the worst case. Indeed, if this function creates a exponentially large number of neighbours,

then it follows that exploring all these neighbours will cost a exponential time. Moreover, in order to guarantee a polynomial delay for a polynomial time computable Children function, the solutions needs to be outputted in a smarter way. In fact, if the algorithm outputs a solution as soon as it encounters it, when dealing with a exponential path on the spanning tree, an exponential length backtrack without any outputs can occur and increase the delay. In order to solve this issue, Takeaki Uno proposed in 2003 two modifications to the enumerations algorithms in order to reduce the delay [32]. The one that is interesting for the Reverse Search is the one called *Alternative Output Method* and more specifically the *Modified Internal Output Algorithm* which consists in outputting before or after the recursive call depending on the evenness of the recursion. This way, half of the solutions will be outputted on the way forward of the recursion and the other half on the backtrack part.

Finally, here is the algorithm template for a Reverse Search algorithm :

Algorithm 1: Reverse Search algorithm template

Input: The polynomially computable Children function to travel on the spanning forest of the solutions supergraph G and $\mathcal{S}$ the set of root nodes of G

Output: All solutions enumerated

```
1: foreach root node S ∈ 𝒮 do
2:     Dive (S, 0)
3: Function Dive (S, depth):
       if depth is even then
4:         Output S
       foreach child X ∈ Children (S) do
5:         Dive (X, depth + 1)
       if depth is odd then
6:         Output S
```

Proximity Search

Recently, in 2019, Alessio Conte and Takeaki Uno introduced in [13] a brand new enumeration framework called *Proximity search* which was defined for maximal induced subgraph enumeration in an other article in 2021 [12]. This technique is similar with the Reverse Search technique as they are both based on the exploration of a solutions supergraph. However, Proximity Search algorithm relies on a specific neighbour function which is computable efficiently and makes the supergraph of solutions strongly connected. Conceiving this neighbouring function may lead to various difficulties to overcome in order to conceive such enumeration algorithm. Indeed, as the number of solutions of an enumeration problem can be exponentially

large as the input size, a solution might have an exponentially large number of neighbours in the solution graph. Moreover, the strongly connected property needs to be verified, otherwise it cannot be guaranteed that all the solutions will be reachable by performing a Depth First Search starting from any solution. In summary, this technique is based on the existence of a neighbouring function that creates a polynomially maximal degree and strongly connected solutions supergraph. However, how does this strongly connected property can be proven ? It comes from the "proximity" notion. Obviously, as the framework is called Proximity Search, it induces that a relation of proximity between two solution exists. Actually, the notion is to be defined below.

Let us consider two distinct solutions S and T defined by a set of elements from $e_1, e_2, ..., e_n$. The notion of proximity between two solutions can be defined as the *longest common prefix* between S and T. It follows that the longer is this prefix, the closer S and T are in the solution graph. To create this proximity measure, one can create an ordering on the vertices for each solution.

Definition 20. *Given two solution S and T, $S\widetilde{\cap}T$ define the proximity between S and T by their longest common prefix.*

Remark 5. *The proximity operation $\widetilde{\cap}$ is not symmetric hence we may have $S\widetilde{\cap}T \neq T\widetilde{\cap}S$.*

Finally, the authors defined the notion of *proximity searchable* which described the existence of a Proximity Search algorithm for a specific enumeration problem P.

Definition 21 (Proximity Searchable)**.** *An enumeration problem P is proximity searchable if there is a proximity function $\widetilde{\cap}$ and a neighbouring* Neighbourhood *function such that a solution of P can be identified in polynomial time in the output size, the neighbouring function is computable in polynomial time in the output size and for every two distinct solutions S and T there is a solution $S' \in$ Neighbourhood(S) such that $|S'\widetilde{\cap}T| > |S\widetilde{\cap}T|$ (S' is "closer" to T than S).*

The last point of this definition induces that for any two solutions S and T there is S' a neighbor of S (from the neighbouring function) which is "closer" to T than S. It follows that consecutively using the neighbouring function and choosing the solution that get closer to T will lead to reach T at some point.

An algorithm created using Proximity Search lead to a polynomial delay algorithm. However, as for the Reverse Search technique, Proximity Search explores the solutions space. And for the same reason as Reverse Search, it is necessary to use the Modified Internal Output Algorithm to ensure the delay. Indeed, the DFS can lead to an exponential backtrack path which will extend the delay. Moreover, Proximity Search does not guarantee a polynomial space algorithm. It can be seen by the fact that all solutions already explored need to be remembered in order to avoid outputting it twice.

Finally, as for the Reverse Search technique, please see below a template for a Proximity Search algorithm :

Algorithm 2: Proximity Search algorithm template

Input: The polynomially computable NEIGHBOURHOOD function to explore the strongly connected solutions supergraph G and S a solution of the problem

Output: All solutions enumerated

```
1: 𝒮 ← ∅ // The set of already found solutions
2: EXPLORE (S, 0)
3: Function EXPLORE (S, depth):
4:     𝒮 ← {𝒮 ∪ S}
       if depth is even then
5:         Output S
       foreach neighbour X ∈ NEIGHBOURHOOD (S) do
6:         EXPLORE (X, depth + 1)
       if depth is odd then
7:         Output S
```

2.2 Potential Maximal Cliques

Showcasing the potential maximal cliques needs some preliminary notions to be introduced too. Indeed, the potential maximal cliques are strongly related to cliques and chordal graphs, defined in Chapter 1. Yet, even with the knowing of cliques and chordal graphs, there is one more key structure that has been studied in parallel with potential maximal cliques in this master thesis : the separators. Indeed, during the potential maximal cliques studies, the separators showed a strong connection to this former [8]. Before defining explicitly the potential maximal cliques, a brief historical review comes first.

It is in the late 90s and more precisely in 1998 that Vincent BOUCHITTÉ and Ioan TODINCA defined, during TODINCA thesis, the notion of *maximal set of neighbour separators* in [6]. At first, it was defined as a tool to compute the treewidth in some specific classes of graphs such as circle and circular arc graphs, chordal bipartite graphs or weakly triangulated graphs. It is only in 1999 that the name *potential maximal clique* emerged with the article [7]. In this paper, the authors claim that if all potential maximal cliques can be listed in polynomial time for some classes of graph, then computing the treewidth and the minimum fill-in is polynomially tractable. This result really pushed forward the potential maximal cliques study. Finally, BOUCHITTÉ and TODINCA designed an output-quadratic algorithm to enumerate all potential maximal cliques in 2002 [8]. Nowadays, these articles are still studied because of the possible applications in various graph theory fields.

The following of this section will be used to first, define the potential maximal cliques along with some of their properties and characterisations, secondly describe

the outline of the first algorithm enumerating them, and finally, exhibit an overview of different outcomes due to potential maximal cliques.

2.2.1 Definition, properties and characterisations

As previously written, potential maximal cliques have various relations to other graph theory objects or classes. Hence, there are multiple characterisations of this object. The most well-known will be presented below.

Definition 22 (Potential Maximal Clique)**.** *A set $\Omega \subseteq V$ is a potential maximal clique of G if Ω is a maximal clique in some minimal triangulation of G.*

We can also define a potential maximal clique using only minimal separators and minimal triangulation as presented in [26].

Lemma 5. *A triangulation of a maximal set of pairwise parallel minimal separators is a minimal triangulation.*

From this lemma, we deduce that transforming a maximal set of pairwise parallel minimal separators into a clique generate at least one potential maximal clique.

Lemma 6. *A set S is an inclusion wise minimal separator if and only if all connected components of $G \setminus S$ are full components associated to S.*

Proof. By definition of S, there is no subset X of S such that X is also a separator. It means that all vertices of S are connected to the same connected components of $G \setminus S$ so they are all full components associated to S. □

Lemma 7 ([8])**.** *If Ω is a potential maximal clique from G, then for any pair of vertices $x, y \in \Omega$, x and y are either adjacent in G or they are connected by a path in a connected component C_i of $G \setminus (\Omega \setminus (\{x, y\}))$. For the second case, we will say that x and y are connected by a virtual-edge via C_i.*

In the rest of the paper, we will use the property from Lemma 7 as *reachability*. We will say that a vertex x has the reachability property to the vertex y if there exists a direct or a virtual-edge between them.

With this previous lemma and the full component property defined before, we can give an other characterisation of potential maximal cliques :

Lemma 8 ([8])**.** *Let $X \subseteq V$ a set of vertices. Ω is a potential maximal clique if $G \setminus \Omega$ has no full component associated to Ω and if for every pair $x, y \in \Omega$, Lemma 7 holds.*

Definition 23 (Active/Inactive separator [8])**.** *Let Ω be a PMC of a graph G and $S \subset \Omega$ a minimal separator of G. S is said to be active for Ω if Ω is not a clique in the graph G where all $S_i \in \Delta(\Omega) \setminus S$ are completed where $\Delta(\Omega)$ is the set of minimal separators included into Ω. Otherwise, S is inactive for Ω.*

2.2.2 Bouchitté and Todinca algorithm

As a first prominent result about potential maximal cliques comes the output-quadratic enumeration algorithm from Vincent Bouchitté and Ioan Todinca [8]. Here, we will outline its functioning and show what can be improved. First, the algorithm concept is based on the finding of the potential maximal cliques in a incremental way manner by finding the potential maximal clique in a subgraph before extending these solutions to a larger subgraph.

In order to guarantee the efficiency of this technique, some properties have been verified. Especially, there are four cases that the algorithm take care of for creating the potential maximal cliques. Let us denote G' the subgraph of $G = G' \cup \{a\}$. Two of the fourth cases are testing if a former potential maximal clique of G' can be extended into one in G by either keeping the exact same potential maximal clique or either adding the vertex a to it. Then, one case is based on the addition of a to any separators of the graph to create a potential maximal clique. Lastly, the final case is taking into account some specific potential maximal cliques which are S-active. Those potential maximal cliques are hard to find and in this algorithm, they found them by searching all the separators T that lie in a connected component of a minimal separator S of G. This step is much slower than the other cases. Hence, it makes the algorithm quadratic in the number of separators, and hence possibly quadratic in the number of potential maximal cliques.

In order to improve this algorithm, the enhancement of the fourth case is probably the best way to go. Indeed, improving this case can, in the best scenario, makes the algorithm become output linear in the number of potential maximal cliques. However, this algorithm has a drawback which cannot be overcome. Certainly, because the algorithm construct the potential maximal cliques from a subgraph, we obtain the outputted solution only when the graph is fully completed, and it follows that all the potential maximal cliques are only outputted at the end of the execution. To get rid of this issue, the only way is to use an other enumeration technique that overcome this drawback.

2.2.3 Research interests and outcoumes

As previously mentioned in the beginning of this chapter, the study of potential maximal cliques is strongly motivated by its link to the treewidth of graphs. Indeed, the enumeration of potential maximal cliques in polynomial time ensures a polynomial time computation of treewidth [8]. Then the treewidth applications are indirectly potential maximal cliques applications. Below, some applications of the treewidth from [4] will be described.

Cholesky factorisation

The Cholesky factorisation (or Cholesky decomposition) is a method to decompose a matrix into a product of matrices. The so created matrices can be very helpful to enhance the efficiency of some calculation. In exemple, they can be used to simulate efficiently the *Monte Carlo simulations* [21] or to accelerate the *quantum chemistry* calculation [3]. The Cholesky factorisation has a connection with treewidth. Indeed, in this factorisation, there are multiple matrices to deal with and it can be shown that bounding those matrices size correspond to bounding the treewidth of a certain graph constructed from the matrices [5].

Expert system

Graph modelling of some kind of expert system have shown a small treewidth in practice. Then, tree-decompositions of such graphs can be done in order to perform efficient certain computation instead of time-consuming statistical computation with uncertainty.

Evolution theory

A very common problem in evolution theory and more precisely in phylogenetic is the Perfect Phylogeny problem. First of all, a *phylogeny* is a tree representation of the evolution history of a *taxa* which is a group of organisms regrouped based on some criteria [16]. As an example of phylogeny, consider Figure 2.6 from [16].

	a	b	c	d	e	f
lamprey	0	0	0	0	0	0
shark	1	1	0	1	0	0
salmon	1	1	1	1	0	0
lizard	1	1	1	0	1	0

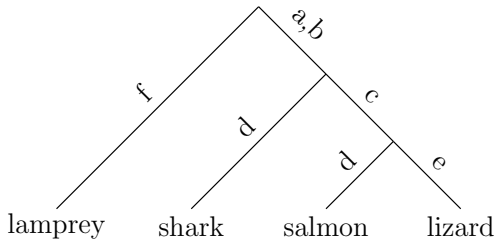


Figure 2.6: A data matrix and a phylogeny. The characters are : (a) paired fins, (b) jaws, (c) large dermal bones, (d) fin rays, (e) lungs, (f) rasping tongue

In Figure 2.6, we can remark that the characters (d) appear in two distinct branches. This phenomena is called *homoplasy* and sometimes cannot be avoided with real data. A phylogeny is called *perfect* if there is no homoplasy.

For the Perfect Phylogeny problem, the input is a binary data matrix with taxa in rows and characters in columns. The objective is to decide if a set of taxa S admit a perfect phylogeny on the characters set C. In optimization, we might want to find the largest set S. It has been shown that the perfect phylogeny problem can be characterized in finding a minimal triangulation in a particular graph. This way, a link with potential maximal arose and an algorithm using it appear in [18].

Chapter 3

Results

When trying to design enumeration algorithms, there is several ways to go. In this internship, I tried different methods to improve the literature algorithm. Indeed, the algorithm proposed by Bouchitté and Todinca has an output polynomial algorithm, quadratic in the size of the output [8]. In fact, as seen in the previous chapters, an output complexity gives no indication in the computation time between two solutions. In this internship, I focused myself on creating another algorithm using either Flashlight Search or Proximity Search to guarantee a delay between two outputted solutions.

In this way, I studied diverse extension problems in order to find if any extension problem that could be used in the potential maximal cliques enumeration is polynomial time solvable. However, we show multiple hardness results of this kind in this chapter. They will be divided as follow. First, as a preliminary, we will introduce the idea of extending a minimal separator, and then a potential maximal clique and finally, the extension of a certain kind of potential maximal clique will be studied.

3.1 Separators extension hardness

As mentioned in the previous chapter, minimal separators have a strong link to potential maximal cliques. This way we hypothesise that being able to enumerate those in an efficient way will help to enumerate the potential maximal cliques. In order to do so, we will first consider the extension of a minimal (a, b)-separator. As a reminder from Chapter 2, solving this extension problem in polynomial time, ensures us the existence of a polynomial Delay algorithm (with polynomial space) to enumerate all minimal (a, b)-separators of a graph. The problem is defined formally below :

MINIMAL (a, b)-SEPARATOR EXTENSION
Input: A graph $G = (V, E)$, a subset $M \subset V$ of *mandatory* vertices, a subset $F \subset V$ of *forbidden* vertices such that $M \cap F = \emptyset$ and two vertices a and b of V.
Question: Is there $X \subseteq V$ such that a minimal (a, b)-separator with $M \subseteq X$ and $X \cap F = \emptyset$?

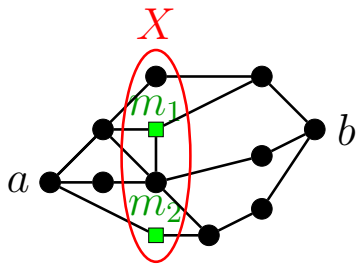


Figure 3.1: An instance and a solution for the MINIMAL (a, b)-SEPARATOR EXTENSION problem

In Figure 3.1, we can see an instance of the problem with $m_1, m_2 \in M$ and $F = \emptyset$. The solution for this instance is the extended set X of M which is a minimal (a, b)-separator. We established some natural properties of separator extension.

Lemma 9 ([8]). *M can be extended if and only if $\exists V' \subset V$, $a, b \in V'$ and $M \subsetneq V'$, M is a minimal (a, b)-separator of $G[V']$.*

Lemma 10. *M is extendable if and only if $\exists T_1, T_2$ such that :*

- *T_1 is a tree rooted in a with $M^+ \supseteq M$ as leaves.*
- *T_2 is a tree rooted in b with $M^+ \subseteq M$ as leaves.*
- *$V(T_1) \cap V(T_2) = M^+$, $V(T_1) \cup V(T_2) = V(G)$ and there is no edge connecting T_1 and T_2.*

Proof. $\Longrightarrow$ direction : If M is extendable, there is a $M^+ \supseteq M$ which is a minimal (a, b)-separator. Then, M^+ admits a full component in $G \backslash M^+$ containing a (denoted by C_a) and another one containing b (denoted by C_b). Consider T_1 as the spanning tree of $G[C_a \cup M^+]$ and T_2 the spanning tree of $G[C_b \cup M^+]$, such that, in both of these spanning tree, all vertices of M^+ are leaves.
$\Longleftarrow$ direction : It is easy to see that the leaves of T_1 and T_2 make a minimal (a, b)-separator which extend M. □

We provide a polynomial reduction from 3-SAT to MINIMAL (a, b)-SEPARATOR EXTENSION.

Theorem 1. *The problem* MINIMAL (a, b)-SEPARATOR EXTENSION *is NP-Complete. Furthermore, under the Exponential Time Hypothesis (ETH),* MINIMAL (a, b)-SEPARATOR EXTENSION *cannot be solved in time $2^{o(n)}$.*

Proof. Obviously, the problem belongs to NP since checking whether a set S is a minimal (a, b)-separator and both contains M and does not intersect F can be done in polynomial time. We will now reduce this problem to the 3-SAT problem. We fix $F = \emptyset$. Let L be the set of literals, x_i and $\overline{x_i}$ the assignations for the literal ℓ_i and let $\Phi = \bigwedge_{i=1}^{m} C_i$ with the clauses $C_i = \ell_j \vee \ell_k \vee \ell_m$ for $\ell_j, \ell_k, \ell_m \in L$ be a CNF[1] formula with n literals and m clauses and we that Φ is satisfiable if and only if there is an extension of M. We will note $\overline{C_i} = \overline{\ell_j} \wedge \overline{\ell_k} \wedge \overline{\ell_m}$ the negation of C_i. This clause is $True$ if and only if C_i is $False$.

Construction. In the following, we will transform a formula Φ into a graph $G(\Phi)$. In order to create an instance of 3-SAT from our problem, we propose a gadget to represent literals. It represents the selection of the value of a literal ℓ_i, with a XOR mechanism. In Figure 3.2, the square and diamond nodes are in the mandatory set

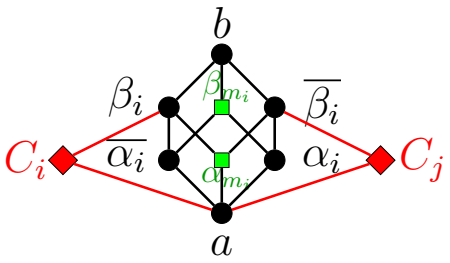


Figure 3.2: The XOR gadget to represent a ℓ_i together with its link to $C_i = \overline{x_i}$ and $C_j = x_i$ (trivial instance)

M, and the other can be used for the expansion of M. Let us call α_i (resp. β_i) the node representing x_i in $N(a)$ (resp. $N(b)$).

Without taking into account the red diamond nodes, we can see that we are forced to take either both α_i and β_i nodes or both $\overline{\alpha_i}$ and $\overline{\beta_i}$ nodes to expend the green square α_{m_i} and β_{m_i} nodes (otherwise the (a, b)-separator will not be minimal).

We complete this gadget by adding the clauses. Each clause C_i is represented by a vertex in M and this vertex will be linked to a and for every literals $\ell_i \in \overline{C_i}$, to the vertex β_i if $\ell_i = x_i$ in C_i (and to $\overline{\beta_i}$ if $\ell_i = \overline{x_i}$ in C_i). This graph can be built in polynomial time since for a formula with n literals and m clauses, we need $6n + m$ nodes and $12n + 4m$ edges.

From 3-SAT to Minimal (a, b)-separator extension. Now we will show that, if there is a truth assignment of the 3-SAT formula, then there is a minimal (a, b)-separator which extends M. Let $A = \{0, 1\}^i$ with $i = |L|$ be a truth assignment of Φ where $A[i]$ is the assignation for x_i in A (1 for $True$ and 0 for $False$). By definition of A, for every clause, there is at least one literal that makes this clause $True$. With our XOR gadget, the only possible extension for a literal ℓ_i is either by taking α_i

[1] Conjunctive Normal Form

and β_i or $\overline{\alpha_i}$ and $\overline{\beta_i}$. We will consider the first option as assigning x_i to $True$ and the second to $False$.

Considering the α_i and β_i as the vertices that make x_i $True$ (resp. $\overline{\alpha_i}$ and $\overline{\beta_i}$ for x_i $False$), we construct our extension by adding the vertices corresponding to A. As the C_i vertices are in M and the only 3 paths from a to b crossing C_i are by also crossing β_i (or $\overline{\beta_i}$) for a $x_i \in \overline{C_i}$, then if we add all those β_i (or $\overline{\beta_i}$) in the extension of M then the separator is not minimal because of C_i. However, as our extension is constructed from A which is a truth assignment, for each C_i, there is at least one x_i from $\overline{C_i}$ such that β_i (or $\overline{\beta_i}$) is not in the separator. In conclusion, the separator constructed is minimal.

From Minimal (a, b)-separator extension to 3-SAT. If there is a minimal (a, b)-separator S extending M, then for each node C_i, there is a $x_i \in \overline{C_i}$ such that its corresponding $\alpha_i, \beta_i \notin S$. Considering the α_i, β_i, $\overline{\alpha_i}$ and $\overline{\beta_i}$ the same way as the previous direction, we construct a truth assignment for Φ by taking the value of the vertices of the extension of M. This assignment is $True$ because we have at least one literal that makes a C_i $True$. □

We can see in Figure 3.3, an example of an instance of 3-SAT with three clauses. This formula is easily satisfiable by the assignment setting x_1, x_2, x_3, x_4, x_5 to $True$. If we take the (a, b)-separator made of the α_i and β_i nodes and the mandatory vertices of M, we see that the (a, b)-separator is minimal.

From now on, we will discuss some properties about this reduction graph that will be helpful for the next section and more precisely for the next hardness proof because they share the same reduction graph.

Lemma 11. *$\forall S \in G(\Phi)$ separator, a and b cannot be in the same connected component is $G \setminus S$.*

Proof. For an a, b-separator it is obvious, but it is also true for any separator of $G(\Phi)$. Indeed, as $N[a] \cup N[b] = G(\Phi)$ and $N[a] \cap N[b] = \emptyset$. Assume that there is a S separator with at least two connected components C and C', with $a, b \in C$ without loss of generality. Then a vertex of C' is adjacent to either a or b which contradicts the fact that S is a separator. □

Corollary 1. *Each vertex of $G(\Phi) \setminus \{a, b\}$ has either a or b in its neighborhood.*

Corollary 2. *Each vertex $v \in V[G(\Phi) \setminus (M \cup \{a, b\})]$ has degree two in $G(\Phi) \setminus M$. Their neighborhood is either a and a vertex in $N(b)$ or b and a vertex in $N(a)$.*

Lemma 12. *Any (a, b)-separator extending M contains, for all i, at least one vertex from both sets : $\{\overline{\alpha_i}, \beta_i\}$; $\{\alpha_i, \overline{\beta_i}\}$.*

Proof. Since the only vertices that can be used to extend M are the α and β nodes, and since there are two disjoints paths from a to b in $G(\Phi) \setminus M$ which are, for all i, $a, \overline{\alpha_i}, \beta_i, b$ and $a, \alpha_i, \overline{\beta_i}, b$, it is easy to see that we need to take in S at least one vertex of each path. □

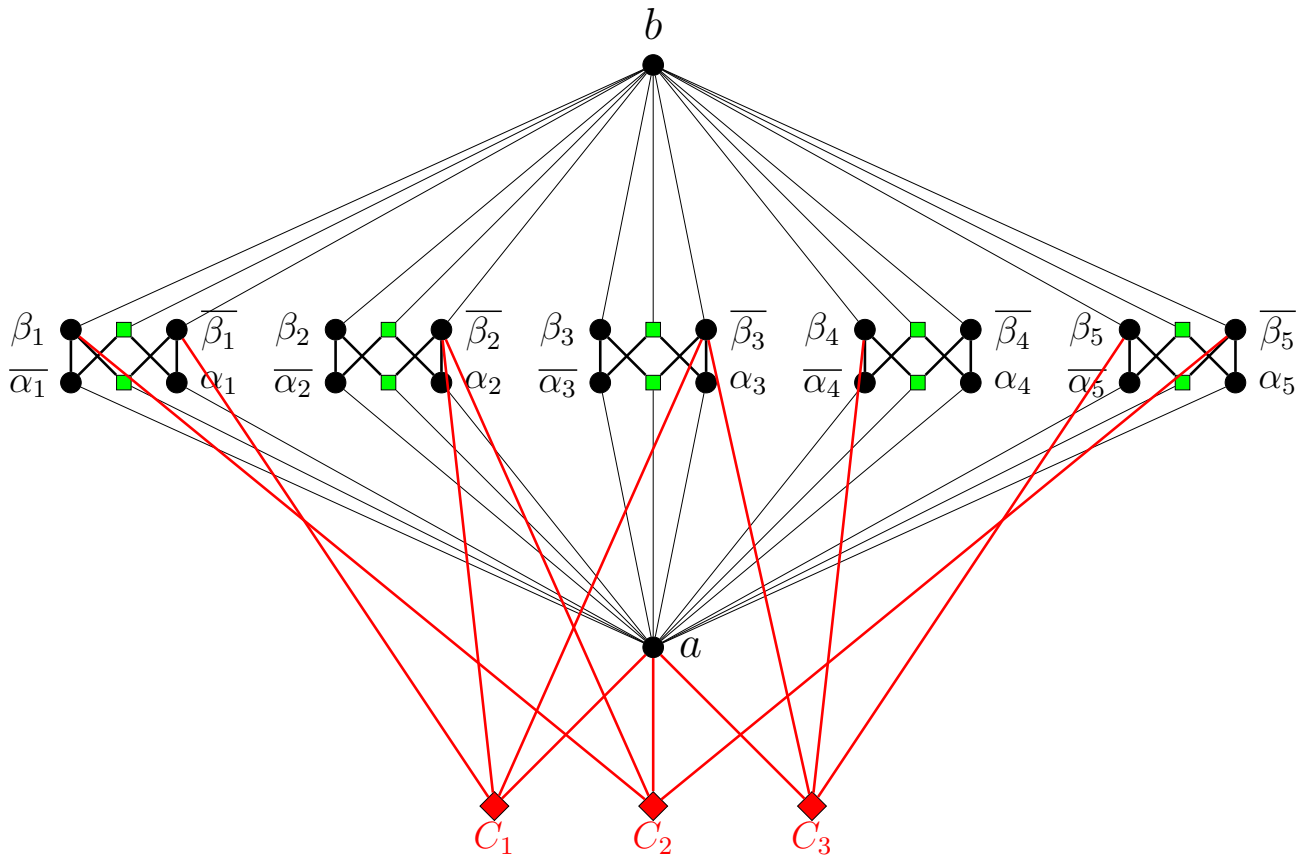


Figure 3.3: An example of the reduction for the 3-SAT formula : $(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_5) \wedge (x_3 \vee \overline{x_4} \vee \overline{x_5})$

Lemma 13. *For any (a,b)-separator S of $G(\Phi)$ extending M and considering C_a and C_b the connected components in $G(\Phi)\setminus S$ containing respectively a and b, $\forall x \in C_a$ (resp. C_b), $\exists v \in S$ such that $N(v) \cap C_a = \{x\}$ (resp. C_b).*

Proof. All vertices from $V[G(\Phi) \setminus (M \cup \{a,b\})]$ have two neighbors in $G(\Phi) \setminus M$ and at most one neighbor in each connected component of $G(\Phi) \setminus S$ (by Corollary 2 and Lemma 12). Furthermore, by the construction of M and its extension, each vertex of C_a or C_b has at least one neighbor in S. In addition, by Corollary 1 and Lemma 12, each vertex of $S \setminus M$ has at most one neighbor in C_a and C_b. □

3.2 Potential maximal cliques extension hardness

From the reduction of minimal (a,b)-separator extension follows a similar NP-Completeness proof for the potential maximal clique extension. Interest ourselves to the study of this problem is very natural since every vertex of the graph is contained in at least one potential maximal clique. The proof will directly follow from the first

one, using the same construction graph and technique. As for the separator, we can define an extension problem for the potential maximal clique :

Potential Maximal Clique Extension
Input: A graph $G = (V, E)$, a subset $M \subset V$ of *mandatory* vertices, a subset $F \subset V$ of *forbidden* vertices such that $M \cap F = \emptyset$.
Question: Does X, a Potential Maximal Clique with $M \subseteq X$ and $X \cap F = \emptyset$ exist ?

Let us note, for all i, $B(C_i)$ the set of β nodes in the neighborhood of the clause neighbour C_i in $G(\Phi)$. The $G(\Phi)$ and the set M is the same as presented for the previous NP-completeness proof, and here, the set $F = \{a, b\}$.

Lemma 14. *If Ω is a potential maximal clique of $G(\Phi)$ extending M and such that $a, b \notin \Omega$, then there is exactly one literal ℓ_i such that $|T_i| = 3$ with $T_i = \{\alpha_i, \overline{\alpha_i}, \beta_i, \overline{\beta_i}\} \cap \Omega$. Otherwise, for all other literals, we have $|T_j| = 2$.*

Proof. We will proceed by case disjunction. From Lemma 11, we deduce that if there is an open path from a to b in $G(\Phi) \setminus \Omega$, then we have only one connected component in $G(\Phi) \setminus \Omega$ which is full to Ω by Corollary 1. Obviously, when $|T_i| = 0$ or 1, there is at least one path from a to b. $|T_i| = 4$ is not possible either because there will be no direct edge or a path via a connected component from α_i to β_i, from $\overline{\alpha_i}$ to $\overline{\beta_i}$, and from α_{m_i} to β_{m_i}. In the following, we will show that all $\forall i, |T_i| = 2$ cannot happen. We exclude the case where we take either $\alpha_i\overline{\beta_i}$ or $\overline{\alpha_i}\beta_i$ because it lead to a path from a to b and by Lemma 11 and Corollary 1, it cannot happen in Ω. Then, there is only three configurations for each ℓ_i to consider :

(1) $T_i = \alpha_i\overline{\alpha_i}$,

(2) $T_i = \beta_i\overline{\beta_i}$

(3) $T_i = \alpha_i\beta_i$ or $\overline{\alpha_i}\overline{\beta_i}$

If all ℓ_i are in the case (1), then we have C_b has a full component associated to Ω. Same for the case (2) and C_a. For the case (3), if there is no C_i such that $B(C_i) \setminus \Omega = \emptyset$, then there are two full components (C_a and C_b). Otherwise there is only one full component C_a. If we mix the case (1) and (2), then we have the β_{m_i} of a ℓ_i in case (1) that can't reach by a virtual-edge via a connected component a $\alpha_{m_{i'}}$ of a $\ell_{i'}$ in case (2). If we blend the case (1) and the case (3), then, we have either 0 full component if there is a C_i such that $B(C_i) \setminus \Omega = \emptyset$ but the C_i node will not be linked by a virtual-edge to any β_{m_i} in the case (1) or, if no such C_i exist, C_b will be a full component associated to Ω. If we mix case (2) and the case (3), we have C_a in full component associated to Ω.
In conclusion, we see that we cannot have $\forall i, |T_i| = 2$. We will show that we cannot have more than one ℓ_i with $|T_i| = 3$. With $|T_i| = 3$, there is a vertex u from T_i that is not connected to C_a and there is a vertex v not connected to C_b. We can easily see that for any pair of $\ell_i, \ell_{i'}$ such that $|T_i| = 3$, the vertex u (resp. v) of ℓ_i does not

have a virtual-edge to the vertex v' (resp u') of $\ell_{i'}$.
At last, if there is exactly one ℓ_i with $|T_i| = 3$, then all other ℓ_i have $|T_i| = 2$. Furthermore, other ℓ_i with $T_i = 2$ cannot be in the case (1) or (2) because, as those cases are full to only one connected component and the ℓ_i with $|T_i| = 3$ is not full to any connected component, a pair of vertices not linked by a virtual-edge via a connected component will exist. In conclusion, all ℓ_i with $|T_i| = 2$ are in the case (3) such that there are two full components which correspond to a minimal (a, b)-separator because, by Lemma 13, adding a third vertex in a ℓ_i with $|T_i| = 2$ will break the full component property of a connected component, so in order to hold property from Lemma 7 have Ω as a potential maximal clique, we need to have two full components before adding it. □

Corollary 3. *Given a potential maximal clique Ω extending M and avoiding a and b, for each gadget ℓ_i of $G(\Phi)$, Ω contains at least either $\alpha_i\beta_i$ or $\overline{\alpha_i}\overline{\beta_i}$.*

Corollary 4. *For Ω a potential maximal clique of $G(\Phi)$ extending M and avoiding a and b, by Lemma 14, all ℓ_i with $|T_i| = 2$ have two full components associated to Ω.*

Lemma 15. *Let Ω be a potential maximal clique of $G(\Phi)$ extending M and avoiding a and b. For all i we have $B(C_i) \setminus \Omega \neq \emptyset$.*

Proof. Let us suppose that such a C_i exists. Then the C_i vertex is only connected to the C_a component. Moreover, we know, by Lemma 14 and Corollary 4, that there is exactly one ℓ_j with $|T_j| = 3$ and that all ℓ_i with $|T_i| = 2$ have two full components. There are two cases to consider. First, the ℓ_j with $|T_j| = 3$ is none of the C_i literals. The second case, is when it is one of the C_i literals. In the first hand, as the C_i node is only connected to C_a, then, it will not be linked by a connected component to the β vertex which is only linked to C_b in the ℓ_j where $|T_j| = 3$. In the other hand, if the ℓ_j with $|T_j| = 3$ is one of the C_i literals, then, we can also decompose in two situation. Either the ℓ_i with $|T_i| = 3$ has all the α nodes and the C_i node will not be linked via a connected component to the β_{m_i} of this ℓ_i or it has all the β node and C_i will not be linked via a connected component to the β vertex which is only linked to C_b. In all cases, it contradicts the fact that Ω is a potential maximal clique, so, such C_i can't exist. □

Remark 6. *Concretely, Lemma 15 means that there is no C_i such that $\{\beta_h, \beta_j, \overline{\beta_k}\} \subseteq \Omega$ knowing that $C_i = \overline{x_h} \cup \overline{x_j} \cup x_k$.*

Theorem 2 (Hardness of PMC extension)**.** *The problem* Potential Maximal Clique Extension *is NP-Complete.*

Proof. In this proof, we will use the graph $G(\Phi)$ described in Theorem 1. We will show that if you are able to solve 3-SAT, then you can extend M in a minimal (a, b)-separator in $G(\Phi)$ and then you can extend this separator into a potential maximal

clique avoiding a and b. We will also show that given a potential maximal clique avoiding a and b, you can find a minimal (a, b)-separator and a truth assignment for a 3-SAT formula.

Construction. For this part, we will keep the graph $G(\Phi)$ from Theorem 1.

From 3-SAT to PMC extension. By Theorem 1, given a truth assignment of Φ we can construct a minimal (a, b)-separator S. Now, we will use S to create the potential maximal clique. From Theorem 4.6 of the article [26] from Parra and Scheffler, it is known that a triangulation is minimal if and only if it is a triangulation of a maximal set of parallel separators (see Lemma 5). In our case, we know the minimal (a, b)-separator S and in that case we can triangulate a maximal set of parallel separators including S and our potential maximal clique will be any maximal clique containing S in the triangulated graph.

In order to make a better understanding of the reduction graph, we propose a proof from the graph. In the first place, if there is a minimal (a, b)-separator $S \in G(\Phi)$, it's mean that S has two full components C_a and C_b (containing respectively a and b). We construct Ω our potential maximal clique by adding any vertex $x \in G \setminus S$. We will note C_x the connected component containing x in $G \setminus S$. C_x is obviously either C_a or C_b. Without loss of generality, let $\overline{C_x}$ the connected component in $G \setminus S$ which does not contain x. In order to have a potential maximal clique, we need to break the full component property but keep the property of the Lemma 7. From the Lemma 13, adding x to S will break the full component property of C_x. Moreover, as S is a separator, x cannot have an edge to the other connected component so the full component property of $\overline{C_x}$ also falls. Secondly, from Lemma 13, x is clearly connected by a direct edge to the vertices which break the full component property of the C_x and to the other by an virtual-edge. The property hold for the vertices of S because $\overline{C_x}$ is a full component associated to S. In conclusion, Ω is a potential maximal clique.

From PMC extension to 3-SAT. Assuming we have a potential maximal clique Ω avoiding a and b, construct the minimal (a, b)-separator S by taking the close separator of C_a or C_b in $G(\Phi) \setminus \Omega$ depending on the number of taken α and β nodes and, by Theorem 1, construct the truth assignment for the formula Φ. Let us note T_α and T_β the number of α nodes (resp. β nodes) in Ω. By Corollary 3, for each ℓ_i there is at least one α node and one β node in Ω and thus, by Lemma 14, there is $T_\alpha > T_\beta$ or $T_\alpha < T_\beta$. Selecting $S = S(C_a)$ if $T_\alpha < T_\beta$ and $S = S(C_b)$ otherwise will ensure having all vertices of M in the separator. Indeed, if $T_\alpha > T_\beta$, then there is a ℓ_i with all the α nodes in Ω and it means that the β_{m_i} node which is mandatory is not connected to C_a. In the other way, the fully covered α nodes on the path from a to b will not be connected to C_b but all α_{m_i} and β_{m_i} nodes will be. Moreover, all C_i nodes are reachable from C_a and by Lemma 15, we ensure that all C_i nodes are connected to C_b. Finally, by definition of a close separator (Definition 17), S is a minimal (a, b)-separator. □

3.3 S-Active potential maximal cliques extension hardness

Finally, we describe our last hardness result about potential maximal cliques enumeration. This last result is considering the creation of a particular kind of potential maximal cliques. Indeed, in the main theorem of [8], Bouchitté and Todinca present four ways to obtain a new potential maximal cliques to enumerate. We noticed that in this algorithm, one type of potential maximal cliques have a slow computation time and forced the algorithm to have a quadratic complexity. Actually, these potential maximal cliques are the one involving an active separator. Indeed, in their algorithm, those potential maximal cliques are found by, given a separator S, trying each separator T inside a full component to create a potential maximal clique. This way, the potential maximal cliques where S is active are found inefficiently, and similarly to the other cases, with the occurrence of redundancy of outputs.

With the hope to improve their algorithm to a output linear one, we studied the enumeration of S-active potential maximal cliques. The S-Active Potential Maximal Clique Extension problem is a sub-problem of Potential Maximal Clique Extension problem and is formally defined as follow :

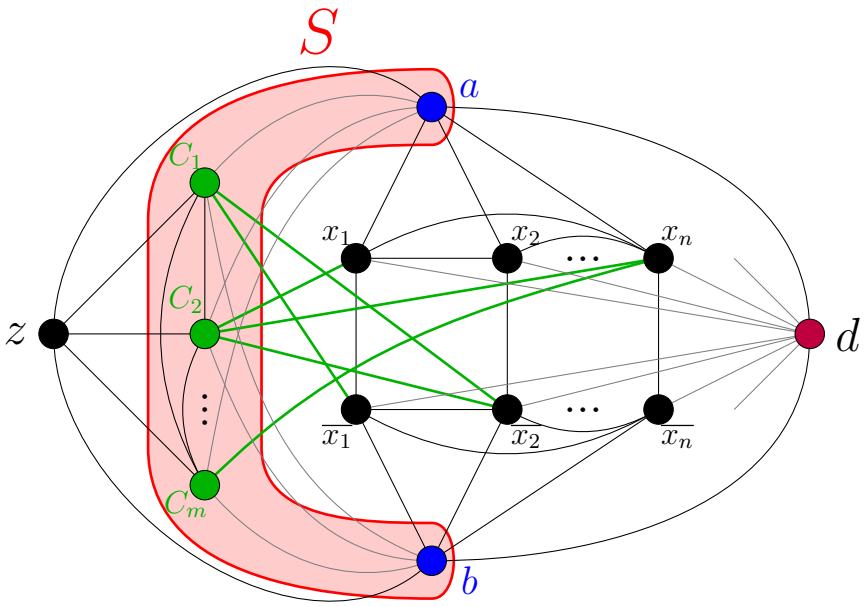
S-Active Potential Maximal Clique Extension
Input: A graph $G = (V, E)$ and a subset $S \subset V$ a minimal separator of G.
Question: Does X, a Potential Maximal Clique with $S \subset X$ and S active for X exist ?

Theorem 3 (Hardness of S-Active PMC extension)**.** *The problem* S-Active Potential Maximal Clique Extension is NP-Complete.

Proof. As the two previous NP-Complete problems, this problem also obviously belongs to NP since we can check if a set is a potential maximal clique and if a pair of non adjacent neighbours appear together in exactly one minimal separator in polynomial time by simply calculating all the minimal separators of the potential maximal clique. In order to prove this statement, a 3-SAT graph construction from S-Active Potential Maximal Clique Extension will be described first. For notations purposes, let us consider Φ the 3-SAT formula with n literals and m clauses.

Construction. First of all, the graph of the reduction must include a separator S and this separator will be active for a potential maximal clique only if there is a satisfiable assignment for the literals. This way, the S has two mirroring full components that represent the same 3-SAT formula such that extending S into a potential maximal clique where S is active will lead to solve the formula. First the separator will be constituted of $m + 2$ vertices, hence one vertex per clause and two vertices a and b. All the vertices of S will be adjacent to each other except for a and b. Secondly, each literal affectation will be represented by a vertex in the full

components. Hence, it will have $2n$ vertices. Each literal and its two affectation vertices makes a path from a to b so passing by first x_i and then $\overline{x_i}$. Moreover, edges are added between all x_i nodes and also between the $\overline{x_i}$ nodes (there is no $x_i\overline{x_j}$ edges with $i \neq j$). Said differently, the subgraph induced by all x_i nodes (resp. $\overline{x_i}$) form a clique. Besides, each clause is linked to the negation of each of their literals. Finally, a vertex d that dominates the connected component with a and b is added. In order to help the understanding of the construction, please consider looking at Figure 3.4.



$$\Phi = C_1 \wedge C_2 \wedge ... \wedge C_m$$
$$C_1 = x_1 \vee x_2$$
$$C_2 = \overline{x_1} \vee x_2 \vee \overline{x_n}$$
$$...$$
$$C_m = \overline{x_n}$$

Figure 3.4: Partial schema of the reduction of S-Active Potential Maximal Clique Extension to 3-SAT. The vertex z represents the other mirrored half

From 3-SAT to S-Active PMC extension. Given a 3-SAT formula Φ, let us consider a truth assignment $A = \{0,1\}^n$ of Φ. Firstly, let us represent A in the reduction graph by taking the corresponding affectation vertices in the potential maximal clique X. X is constructed from the union of S, the A vertices and the dominating vertex d. The last is obviously in the potential maximal clique because of its dominating property. Indeed, to separate a from b, we obviously need d. By definition of a truth assignment, all clauses have at least one literal satisfied. It

follows that, by the construction of the graph, there is at least one neighbour of each clause that does not belong to X. Furthermore, as for any i, all x_i nodes are linked together and all $\overline{x_i}$ also, and, as the definition of a truth assignment implies that, for any i, either x_i or $\overline{x_i}$ is chosen, witness that each taken vertices (A vertices) for the potential maximal clique will be able to reach all other vertices of the connected component and S (either directly or by its negation vertices). Moreover, as S has two full components, it follows that S alongside the A vertices verify Lemma 7. Lastly, we need to ensure that there is no full components. To begin with, we can notice that any pair of not taken affectation vertices with different values will not be in the same connected component as their negation in $G \setminus X$. Furthermore, vertex a does not have any link into the connected component of $G \setminus X$ involving false affectation. The converse with b and true affectation hold. Finally, there is no full components and by Lemma 8, X is a potential maximal clique.

In conclusion, a and b will not be linked to the same separators of the connected components of $G \setminus X$. From that last conclusion follows that the A vertices and d construct a minimal a, b-separator. As a and b are not in the same separators of $G \setminus X$ except one, it follows that S is active for X with the pair a, b.

From S-Active PMC extension to 3-SAT. For the other side, let us consider X the potential maximal clique. We construct a truth assignment A of Φ by assigning the literals to their corresponding taken affectation vertex in X. As X is a potential maximal clique where S is active and as the only non adjacent pair of vertices of S is a, b, then a and b are connected by exactly one full component of $G \setminus X$. In order to separate a and b, all disjoint paths from a to b must admit a vertex in X. This way, for any i, either x_i or $\overline{x_i}$ node must be in X. Furthermore, there is the d vertex which is mandatory to separate a from b. Finally, X cannot choose at random the affectation nodes because we need to ensure that all clauses have a neighbour not taken in X or the d vertex will not be linked by a virtual-edge to a C_j. In conclusion, the selected affectation nodes in X correspond to a truth assignment A of Φ. □

3.4 Efficient algorithm for P_5-free graphs

In this section, we provide a small result concerning the enumeration of the potential maximal cliques in the P_5-free graphs. This result directly follows from the understanding of some basic property of the P_5-free graphs and from a result from Lokshtanov *et al.* in [24] who showed a polynomial time algorithm for Max Independent Set using the potential maximal clique framework. Indeed, in this article, some properties of potential maximal cliques applied to P_5-free emerge. For instance, one property is of interest for our purpose. As said in Chapter 2, there is a way to improve Bouchitté and Todinca algorithm by simplifying the $S \cup T$ (S minimal separator of G and T minimal separator of a connected component of $G \setminus S$) case. In fact, in the paper, the **Lemma 3.13** states that all these potential maximal

cliques can be found in P_5-free graphs by adding the neighbourhood of any vertex of the separator S to the potential maximal clique. This way, with $|\Delta_G|$ the number of minimal separators of G there is a maximum of $|\Delta_G|n^3$ potential maximal cliques of this kind. It follows from the fact that we need to test all the combinations of S in G, connected component C of $G \setminus S$ and a vertex u of S. To enumerate them, LOKSHTANOV proved a $O(|\Delta_G|n^4)$ bound to the time complexity of those potential maximal cliques calculation since the operation $S \cup (N(u) \cap C)$ can be computed trivially in $O(n)$.

From this result, we can assume that calculating potential maximal cliques in P_5-free graphs is possible in output linear time $O(n^2m|\Delta_G|)$ by modifying the fourth case of TODINCA's algorithm with LOKSHTANOV proposition.

Theorem 4. *Potential maximal cliques can be enumerated in time $|\Delta_G|n^2$ in P_5-free graphs.*

Proof. For each induction graph $G_i = G[v_i]$, instead of computing all pairs S, T of separators, we compute all triplets (S, C, u) where S is a separator, C a full component, and $u \in S$ in $|\Delta_G|n^{O(1)}$. By LOKSHTANOV proposition, all potential maximal cliques of P_5-free graphs are of the form $S \cup (N(u) \cap C)$. For every triplets, we just test if it gives a potential maximal cliques in $O(nm)$. □

Conclusion

During this five months internship, I studied the Potential Maximal Clique object in order to conceive an efficient algorithm to enumerate them all. This subject lead me to discover the enumeration field and more specifically the output sensitive enumeration field. Indeed, in this internship, by trying to design algorithm techniques, I finally found different hardness results showing the impossibility of such techniques to be used for the enumeration of potential maximal cliques. Those results will thus be helpful to pursue the search of an efficient enumeration algorithm since they indicate that some enumeration techniques are not relevant. In particular, we showed the difficulty of using Flashlight Search by the hardness of several extension problems. However, it does not prove that Flashlight Search cannot be used to enumerate the potential maximal cliques. Indeed, one can possibly find a new specific extension problem which is polynomial time solvable to enumerate potential maximal clique in polynomial delay and polynomial space using Flashlight Search.

In addition, there are other enumeration techniques that may help us create the algorithm. We can, for example, cite Proximity Search, which is a recent one. This technique can ensure a polynomial delay for the enumeration if a polynomial time computable neighbouring function between two potential maximal cliques can be found.

In conclusion, there are a lot of perspectives to continue the search of such algorithm, and I will continue to search it during a PhD thesis. This PhD thesis will take place at LIMOS directed by both the supervisors of this internship : Vincent Limouzy and Pierre Bergé. In this thesis, I will continue my internship work and I will encounter many other enumeration problem to work on.

Bibliography

[1] Alexander Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*, volume 24. Springer, 2003.

[2] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46, March 1996.

[3] Nelson H. F. Beebe and Jan Linderberg. Simplifications in the generation and transformation of two-electron integrals in molecular calculations. *International Journal of Quantum Chemistry*, 12(4):683–705, 1977. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/qua.560120408.

[4] Hans L. Bodlaender. A tourist guide through treewidth. *Acta cybernetica*, 11(1-2):1–21, 1993. Publisher: University of Szeged.

[5] Hans L. Bodlaender, John R. Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, and minimum elimination tree height. In Gunther Schmidt and Rudolf Berghammer, editors, *Graph-Theoretic Concepts in Computer Science*, pages 1–12, Berlin, Heidelberg, 1992. Springer.

[6] Vincent Bouchitté and Ioan Todinca. Minimal Triangulations for Graphs with "Few" Minimal Separators. In Gianfranco Bilardi, Giuseppe F. Italiano, Andrea Pietracaprina, and Geppino Pucci, editors, *Algorithms — ESA' 98*, pages 344–355, Berlin, Heidelberg, 1998. Springer.

[7] Vincent Bouchitté and Ioan Todinca. Treewidth and Minimum Fill-In of Weakly Triangulated Graphs. In Christoph Meinel and Sophie Tison, editors, *STACS 99*, pages 197–206, Berlin, Heidelberg, 1999. Springer.

[8] Vincent Bouchitté and Ioan Todinca. Listing all potential maximal cliques of a graph. *Theoretical Computer Science*, 276(1):17–32, April 2002.

[9] Caroline Brosse, Oscar Defrain, Kazuhiro Kurita, Vincent Limouzy, Takeaki Uno, and Kunihiro Wasa. On the hardness of inclusion-wise minimal separators enumeration. *Information Processing Letters*, 185:106469, March 2024.

[10] Florent Capelli and Yann Strozecki. Enumerating models of DNF faster: breaking the dependency on the formula size. *Discrete Applied Mathematics*, 303:203–215, November 2021. arXiv:1810.04006 [cs].

[11] Florent Capelli and Yann Strozecki. From Amortized to Worst Case Delay in Enumeration Algorithms, September 2024. arXiv:2108.10208 [cs].

[12] Alessio Conte, Andrea Marino, Roberto Grossi, Takeaki Uno, and Luca Versari. Proximity Search For Maximal Subgraph Enumeration, August 2021. arXiv:1912.13446 [cs].

[13] Alessio Conte and Takeaki Uno. New polynomial delay bounds for maximal subgraph enumeration by proximity search. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 1179–1190, New York, NY, USA, June 2019. Association for Computing Machinery.

[14] de Ridder H.N. et al. Information System on Graph Classes and their Inclusions, 1999.

[15] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1736.

[16] David Fernández-Baca. The Perfect Phylogeny Problem. In Xiu Zhen Cheng and Ding-Zhu Du, editors, *Steiner Trees in Industry*, pages 203–234. Springer US, Boston, MA, 2001.

[17] Serge Gaspers and Simon Mackenzie. On the number of minimal separators in graphs. *Journal of Graph Theory*, 87(4):653–659, 2018. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jgt.22179.

[18] Rob Gysel. Potential Maximal Clique Algorithms for Perfect Phylogeny Problems, March 2013. arXiv:1303.3931 [cs].

[19] Pinar Heggernes. Treewidth, partial k-trees, and chordal graphs.

[20] Pinar Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, February 2006.

[21] Guoqing Huang, Haili Liao, and Mingshui Li. New formulation of Cholesky decomposition and applications in stochastic simulation. *Probabilistic Engineering Mechanics*, 34:40–47, October 2013.

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

[23] Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the Enumeration of Minimal Dominating Sets and Related Notions. *SIAM Journal on Discrete Mathematics*, 28(4):1916–1929, January 2014. Publisher: Society for Industrial and Applied Mathematics.

[24] Daniel Lokshtanov, Martin Vatshelle, and Yngve Villanger. Independent Set in P_5 -Free Graphs in Polynomial Time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 570–581. Society for Industrial and Applied Mathematics, January 2014.

[25] Martín Muñoz and Cristian Riveros. Constant-delay enumeration for SLP-compressed documents. *Logical Methods in Computer Science*, Volume 21, Issue 1, February 2025. Publisher: Episciences.org.

[26] Andreas Parra and Petra Scheffler. Characterizations and algorithmic applications of chordal graph embeddings. *Discrete Applied Mathematics*, 79(1):171–188, November 1997.

[27] R. C. Read and R. E. Tarjan. Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees. *Networks*, 5(3):237–252, 1975. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.1975.5.3.237.

[28] Donald J. Rose, R. Endre Tarjan, and George S. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on Computing*, 5(2):266–283, June 1976. Publisher: Society for Industrial and Applied Mathematics.

[29] Jean-François Rual, Kavitha Venkatesan, Tong Hao, Tomoko Hirozane-Kishikawa, Amélie Dricot, Ning Li, Gabriel F. Berriz, Francis D. Gibbons, Matija Dreze, Nono Ayivi-Guedehoussou, Niels Klitgord, Christophe Simon, Mike Boxem, Stuart Milstein, Jennifer Rosenberg, Debra S. Goldberg, Lan V. Zhang, Sharyl L. Wong, Giovanni Franklin, Siming Li, Joanna S. Albala, Janghoo Lim, Carlene Fraughton, Estelle Llamosas, Sebiha Cevik, Camille Bex, Philippe Lamesch, Robert S. Sikorski, Jean Vandenhaute, Huda Y. Zoghbi, Alex Smolyar, Stephanie Bosak, Reynaldo Sequerra, Lynn Doucette-Stamm, Michael E. Cusick, David E. Hill, Frederick P. Roth, and Marc Vidal. Towards a proteome-scale map of the human protein–protein interaction network. *Nature*, 437(7062):1173–1178, October 2005. Publisher: Nature Publishing Group.

[30] Jeremy P. Spinrad. *Efficient Graph Representations.: The Fields Institute for Research in Mathematical Sciences.* American Mathematical Soc., January 2003. Google-Books-ID: RrtXSKMAmWgC.

[31] Ken Takata. Space-optimal, backtracking algorithms to list the minimal vertex separators of a graph. *Discrete Applied Mathematics*, 158(15):1660–1667, August 2010.

[32] Takeaki Uno. Two General Methods to Reduce Delay and Change of Enumeration Algorithms. *NII Technical Reports*, May 2003.