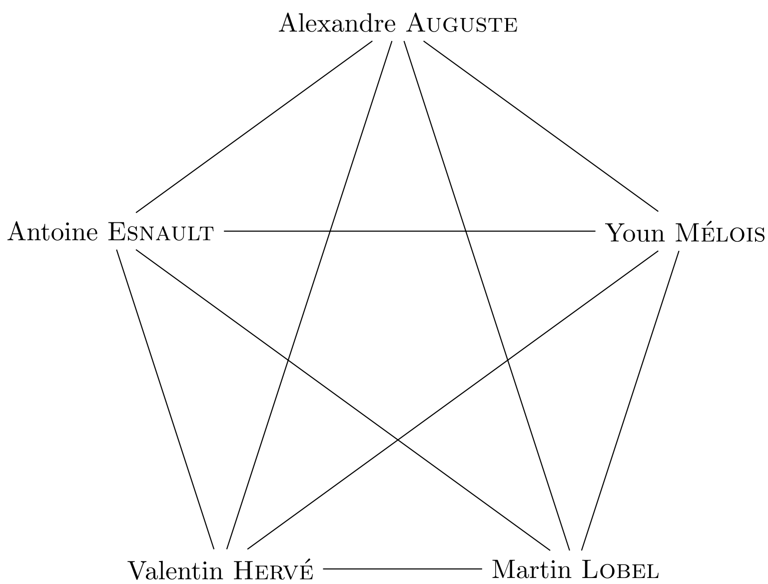
Final Project
The Maximum Edge Weight Clique Problem



Abstract
The Maximum Edge Weight Clique (MEWC) problem is an optimization problem in graph theory that asks for the clique (a subset of vertices, all adjacent to one another) with the maximum total weight in an edge-weighted undirected graph. In the MEWC problem, each edge has a weight, and the weight of a clique is the sum of the weights of its edges. The goal is to find a clique with the maximum possible weight.

Class group:
CIR3 - Team 1

Teacher:
Leandro MONTERO

January 8, 2023

Contents

1 Introduction

1.1 Presentation of the subject

Graph theory is a branch of mathematics that deals with the study of graphs, which are mathematical structures used to model pairwise relationships between objects. Graphs consist of vertices (also called nodes) that are connected by edges. The edges can be either directed (one-way) or undirected (two-way) and can also have a weight.

The Maximum Edge Weight Clique (MEWC) problem is an optimization problem in graph theory that asks for the clique (a subset of vertices, all adjacent to one another) with the maximum total weight in an edge-weighted undirected graph. In the MEWC problem, each edge has a weight, and the weight of a clique is the sum of the weights of its edges. The goal is to find a clique with the maximum possible weight.

Now, the MEWC problem is **NP-hard**, which means that it is not possible to find an efficient algorithm to solve it in polynomial time or that this problem is at least as hard as the hardest problems in NP. It is also the generalization of the Maximum Clique Problem (MCP), which is the special case where all edges have the same weight.

For example, the following graph $G = (V, E)$ has for its set of vertices $V = \{1, 2, 3, 4, 5, 6\}$ and for its set of edges $E = \{(1, 2), (1, 5), (2, 3), (2, 5), (3, 4), (4, 5), (4, 6)\}$. As we can see, the red edges form a clique of size 3 and the other colored edges are each cliques of size 1. We can also easily deduct that the maximum clique of G is the red clique of size 3.

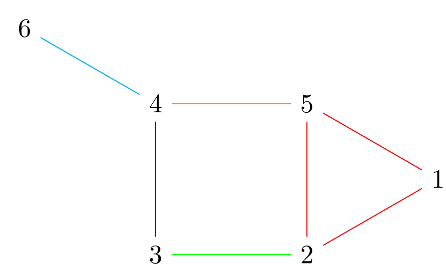


Figure 1: Basic graph example

The MEWC problem can be used to model various types of real-world situations where the goal is to find a subset of objects with the maximum total weight, and the objects are connected by weighted edges. Here are a few examples of such situations:

- **Network design**: In a communication network, the MEWC problem can be used to find the optimal subset of devices (vertices) to include in the network, such that the total cost of communication between the devices (edges) is maximum.
- **Protein interaction**: In biology, the MEWC problem can be used to find the optimal subset of proteins (vertices) in a protein-protein interaction network, such that the total interaction strength (edges) between the proteins is maximum.
- **Social network analysis**: In a social network, the MEWC problem can be used to find the optimal subset of individuals (vertices) with the maximum total relationship strength (edges) between them.

1.2 Configuration

- Ajouter les configurations que l'on a utilise pour les tests du style la config

On this project, we decide to use **the C++ language** to developp and implement our algorithms. Nevertheless, a debate took place, especially on the choice of the language. We hesitated between Python and C++. The first one was for us easier to handle and was a tool in which we had more confidence in our ability to use it efficiently. The latter was nevertheless preferred because of its speed of execution, which was an important criterion for the study. We did have some problems with memory allocation, which made us regret this choice at times.

To share the code between us, we used **Github**[1]. It is a web-based platform for version control, collaboration, and sharing of code, as well as a community of developers who contribute to open source projects and share their knowledge. It was a tool that was difficult for some to get used to quickly, especially on the configuration of the project at home, but which brought us a significant gain in efficiency once we had understood how to use it. To share information and communicate between us, we used **Discord**.

1.3 Example of real-life situations

As we said, the MEWC has many real-life applications in various fields such as social networks, chemistry, bioinformatics. We could model this problem with some fast example like these :

- We can model this problem on social **social networks**, indeed we can represent each **users** as a **vertex** in a graph and add an **edge** between two vertices if the corresponding individuals **have some kinds of relationships together**. The weight of the edge could represent **the degree of relationships** between the two users. The goal here is to identify the group of individuals with the strongest connections within a social network. Some example can be Twitter or Netflix.
- Furthermore, we can model the MEWC problem on **chemistry**, indeed we can represeant each **chemical compounds** as a **vertex** in a graph and add an **edge** between two vertices if the corresponding chemical compounds **have an intermolecular interaction between them**. The weight of the edge could represent the **he strength of the corresponding intermolecular interaction**. The goal here is to identify the compound with the strongest intermolecular interactions from a set of potential drug candidates. Some example can be found on research.

We will take a practical example which could include and involve ISEN students in our future. We will reuse the presented case to illustrate the different algorithms that we will implement later in the report.

An example of real-life situations that can be modelled as MEWC is the team formation process during a project, or in the search for a particular social group.

[1] `https://github.com/sehnryr/Final-Graph-Project-ISEN-CIR3`

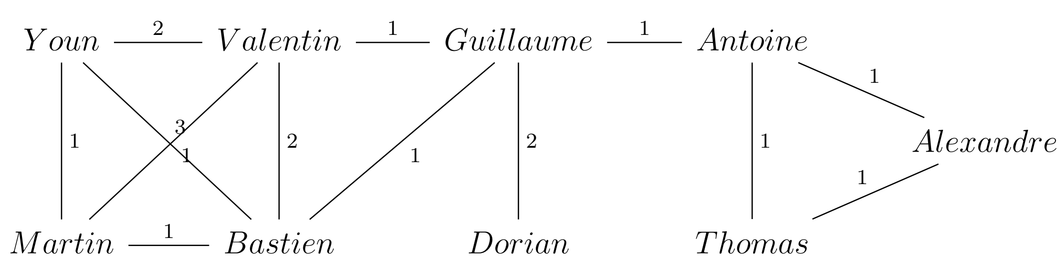
We can imagine that the Student Office of ISEN Nantes is looking to reinforce the video games club of its school. Indeed, the latter has no succession for the following year and is thus led to die if no member presents himself. The future members of the office will have to be in contact with each other during a whole year and it is thus important to find people with common interests so that no tension is formed during their studies. The office has access to the Steam profiles of the students within ISEN (Steam is a video game digital distribution service that gives information about the games played by each one) as well as a record made by the gaming club of the different games played by their members. The fact of playing games in common could bring some people closer, and this makes it a good criterion to create a group that could take over the club because it would share common interests. Otherwise, it would allow to see which games and which group could be present at different events they could organize. Here, forming a team from a group of individuals can be considered as a maximum edge weight clique problem because the goal is to select a subset of individuals such that the members share a common interest.

To model this problem as a maximum edge weight clique problem, we can represent each **individual** as a **vertex** in a graph and add an **edge** between two vertices if the corresponding individuals **share atleast one game**. The weight of the edge could represent **the number of game** that they have in common.

For example, on a small scale we could imagine :

Students	**Game played**
Youn	Minecraft, Civilisations, Lost ARK, Among US
Martin	Minecraft
Valentin	Genshin, Minecraft, Civilisations
Bastien	Genshin, Minecraft, Lost ARK, Among US
Guillaume	CSGO, Genshin, Overwatch, Stardew Valley
Dorian	CSGO, Paladins, Overwatch
Antoine	League of Legends, Stardew Valley
Thomas	League of Legends, The Last of US
Alexandre	League of Legends, Dofus

Which would give us this graph :



The goal of the maximum edge weight clique problem in this context would be to find a complete subset of individuals such that the sum of the weights of the edges between the individuals is maximized. In this example, the maximum weight clique would be the clique consisting of nodes Youn, Valentin, Martin, Maxence with a total weight of $10(2+1+1+1+3+2)$.

2 Exact Algorithm

2.1 Presentation

An exact algorithm for solving the MEWC problem works by exploring all potential cliques in the graph and selecting the clique with the maximum weight. To do this, the algorithm uses a recursive function to explore all possible subsets of V. For each subset, the algorithm computes the weight of the clique formed by the vertices in the subset. If the weight is greater than the current maximum weight, the clique is selected as the current maximum. This process is repeated until all possible cliques have been explored, at which point the algorithm returns the clique the the maximum weight.

The time complexity of this approach is $\mathcal{O}(n^2 \times 2^n)$, where n is the number of vertices in the graph. This is due to the fact that there are 2^n possible subsets of V, and the algorithm must compute the weight of the clique, which by itself has a complexity of $\mathcal{O}(n^2)$ because there are at most $\frac{n(n-1)}{2}$ edges in a complete graph, and compare it to the current maximum weight. Therefore, the algorithm is only feasible for small-scale graphs.

However, there is a better algorithm called the **Bron-Kerbosch** algorithm[1] that can find all maximal cliques in a graph with a time complexity of $\mathcal{O}(3^{\frac{n}{3}})$, where n is the number of vertices in the graph. This algorithm is optimal, as it has been proven by J. W. Moon & L. Moser in 1965[2] that there are at most $3^{\frac{n}{3}}$ maximal cliques in any n-vertex graph.

We can use the Bron-Kerbosch algorithm to find all maximal cliques in the graph, and then compute the weight of each clique. This approach has a time complexity of $\mathcal{O}(n^2 \times 3^{\frac{n}{3}})$, which is much better than the previous approach. However, this algorithm is still not feasible for large-scale graphs.

2.2 How it works

As said before, our algorithm first uses Bron-Kerbosc to obtain all the maximal cliques of the input graph. Then, it looks for which cliques have the highest weight. So we will explain how it does to get the different maximal cliques.

The Bron-Kerbosch pivot algorithm that we used is a more efficient variant of the Bron-Kerbosch algorithm that is used to find all the cliques in a graph. It works in a similar way to the original Bron-Kerbosch algorithm, but uses a pivot vertex to guide the search for cliques.

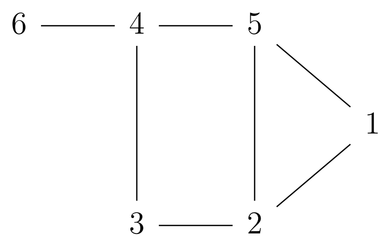
At each step of the algorithm, the pivot algorithm keeps track of three groups of vertices: the candidates, which are the set P of vertices that could potentially be part of the clique, the already-selected vertices, which are the set R of vertices that are definitely part of the clique, and the set X of vertices that have been considered but not selected.

The algorithm selects a pivot vertex from the set of candidates and already-selected vertices, and then adds to the candidate set any vertices that are connected to all of the vertices in the already-selected set, as well as the pivot vertex. It then removes from the candidate set any vertices that are not connected to all of the vertices in the already-selected set, and adds those vertices to the set X of vertices that have been considered but not selected.

This process is repeated until the candidate set is empty. At that point, all of the vertices in the already-selected set form a clique. The algorithm is then run again on the remaining candidates and already-selected vertices to find any additional cliques.

We will now illustrate it step by step with a practical example :

Step 0 :

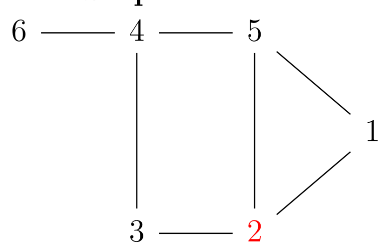


We have here our graph as an example. As said before, the Bron-Kerbosh pivot algorithm will take 3 parameters. R, the set of vertices already selected. P, the set of candidate vertices. X, the set of vertices that have been considered but not selected.

Here, we have :

$$\boxed{R = \{\} \quad P = \{1, 2, 3, 4, 5, 6\} \quad X = \{\}}$$

Step 1 :

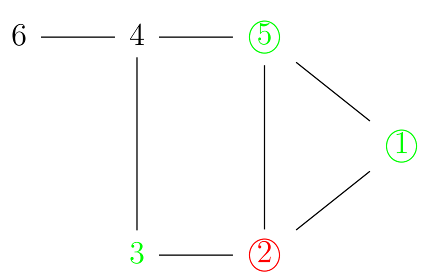


Now, to begin the algorithm, we will need to chose a pivot u. The pivot u should be chosen as one of the degree-three vertices, to minimize the number of recursive calls. Now, we suppose that u is chosen to be vertex 2. We see, that there is 2 vertices that are not adjacent to 2 which are 4 and 5.

Then we know that we will work by starting with these 3 configurations (we know the first, but we don't know yet for the vertex 4 and 6 because some vertex could be added to X or removed from P when we process on 2 or 4) :

$$\boxed{\begin{array}{l} R = \{2\} \quad P = \{1, 3, 5\} \quad X = \{\} \\ R = \{4\} \\ R = \{6\} \end{array}}$$

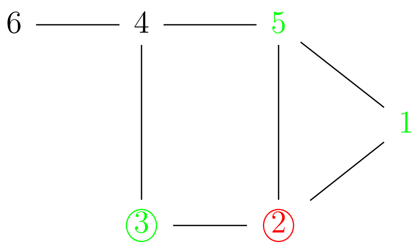
Step 2 :



The algorithm begins by looking at vertex 2 in the graph. It makes a recursive call with using vertex 2 as a starting point. In this recursive call, the algorithm looks at the first group of vertices it was given (vertices 1 and 5) and selects one of them as the pivot vertex. Let's say it selects vertex 1. It then makes a first second-level recursive call for vertex 5. That will eventually find the clique $(1, 2, 5)$.

The recursive call process like this :

$$\begin{array}{llll} R=\{2\} & P=\{1,3,5\} & X=\{\} & \\ R=\{2,5\} & P=\{1\} & X=\{\} & \rightarrow P=\emptyset \text{ and } X=\emptyset \text{ then } (2,5,1) \text{ is a clique.} \\ R=\{2,5,1\} & P=\{\} & X=\{\} & \end{array}$$

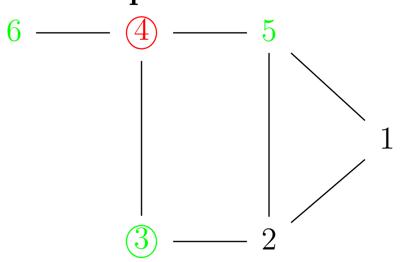


The algorithm then makes a second second-level recursive calls for vertex 3. That will eventually find the clique $(2,3)$. After these two second level recursive calls have completed, vertex 2 is added to X and removed from P.

The recursive call process like this :

$$\begin{array}{llll} R=\{2\} & P=\{1,3,5\} & X=\{\} & \\ R=\{2,3\} & P=\{\} & X=\{\} & \rightarrow P=\emptyset \text{ and } X=\emptyset \text{ then } (2,3) \text{ is a clique.} \\ R=\{\} & P=\{1,3,4,5,6\} & X=\{2\} & \end{array}$$

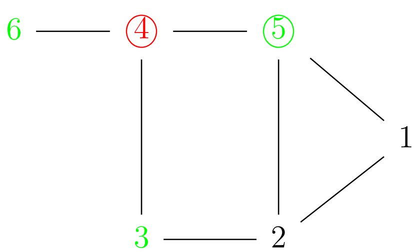
Step 3 :



it will now do an iteration taking 4 as vertices. We note that the vertex 2 belongs to the set X in the outer call to the algorithm, but it is not a neighbor of the vertex 4 and is excluded from the subset of X passed to the recursive call. The algorithm makes a first second-level recursive call for vertex 3. That will eventually find the clique (3, 4).

The recursive call process like this :

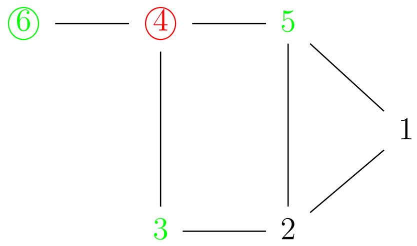
$$\begin{array}{llll} R=\{4\} & P=\{3,5,6\} & X=\{\} & \rightarrow P=\emptyset \text{ and } X=\emptyset \text{ then } (4,3) \text{ is a clique.} \\ R=\{4,3\} & P=\{\} & X=\{\} & \end{array}$$



The algorithm makes a second second-level recursive call for vertex 5. That will eventually find the clique $(4,5)$.

The recursive call process like this :

$$\begin{array}{llll} R=\{4\} & P=\{3,5,6\} & X=\{\} & \rightarrow P=\emptyset \text{ and } X=\emptyset \text{ then } (4,5) \text{ is a clique.} \\ R=\{4,5\} & P=\{\} & X=\{\} & \end{array}$$

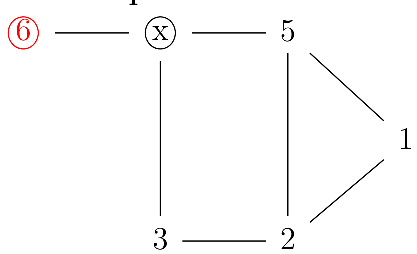


The algorithm makes a third second-level recursive call for vertex 6. That will eventually find the clique $(4, 6)$. After these three second level recursive calls have completed, vertex 4 is added to X and removed from P.

The recursive call process like this :

$R = \{4\}$	$P = \{3, 5, 6\}$	$X = \{\}$	
$R = \{4, 6\}$	$P = \{\}$	$X = \{\}$	$\rightarrow P = \emptyset$ and $X = \emptyset$ then $(4, 6)$ is a clique.
$R = \{\}$	$P = \{1, 3, 5, 6\}$	$X = \{2, 4\}$	

Step 4 :



It will now do a third and final iteration taking 6 as vertex. It makes a recursive call but because P is empty and X is nom empty (the vertex 4 was added to X and removed from P in step 3). Because of that, the algorithm immediately stops searching for cliques and backtracks, because there can be no maximal clique that includes the vertex 6 and excludes the vertex 4.

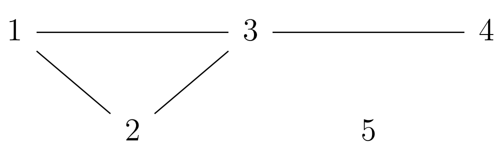
The recursive call process like this :

$$\boxed{R = \{6\} \quad P = \{\} \quad X = \{4\} \;\rightarrow\; \text{No clique was found}}$$

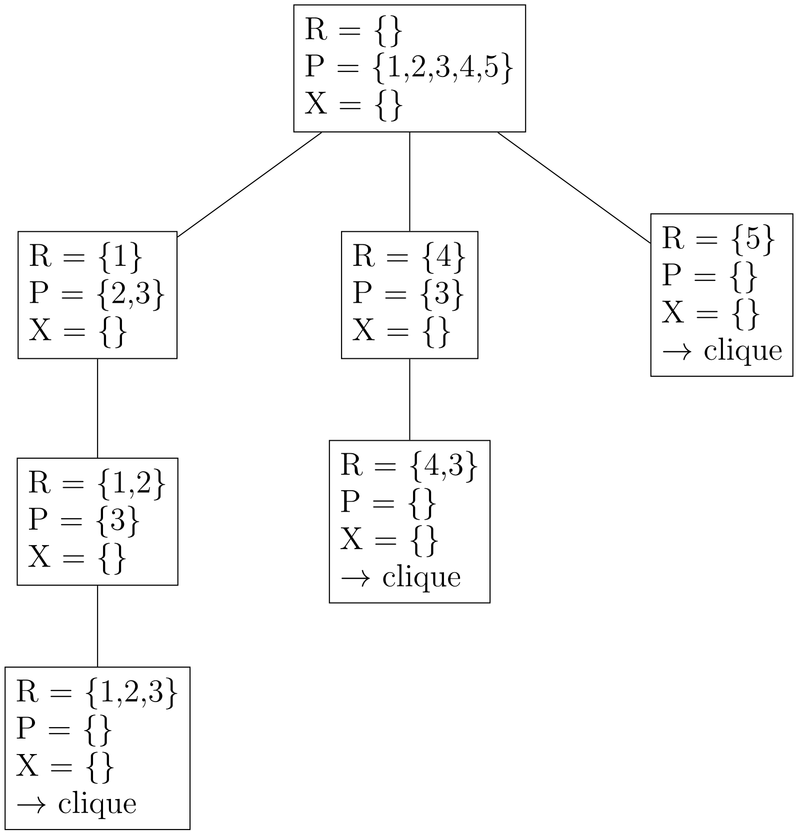
The algorithm is therefore **finished**, and we have obtained the cliques :

$$\{(2, 3), (1, 2, 5), (3, 4), (4, 5), (5, 6)\}$$

To summarize things and try to make things even clearer, we will take another example and make the call tree of it (the precedent example is too big) :



His call tree :



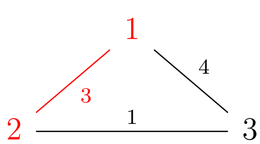
After getting every clique, the exact algorithm will iterate all maximal clique and apply a function that will return the total weight of it.

This function work by iterating every possible pairs of vertices in the clique and the weight of it if there is an edge between them in a variable that start at 0. The variable will be returned as the total weight of the clique.

Some quick example of it :

Step 1 :

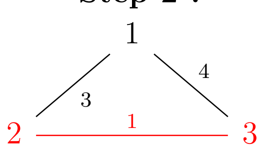
Total Weight = 0 (start)



The function will take the vertex 1 and 2, the edge between them have a weight of 3. It will add 3 in the variable "Total Weight".

Total Weight $= 0 + 3 = 3$

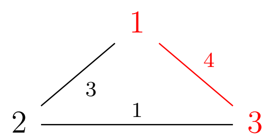
Step 2 :



The function will take the vertex 2 and 3, the edge between them have a weight of 1. It will add 1 in the variable "Total Weight".

Total Weight $= 3 + 1 = 4$

Step 3 :



The function will take the vertex 1 and 3, the edge between them have a weight of 4. It will add 4 in the variable "Total Weight".

$$\text{Total Weight} = 4 + 4 = 8$$

The total weight of this clique is 8.

The algorithm then takes the clique with the greatest weight. And **the MEWC is solved**.

2.3 Pseudo code

2.4 Complexity

2.5 Bad Instance

The exact algorithm have no bad instance, he will always find the optimal solution each time. However, it will take a fairly long time to do so.

2.6 Experiments

2.7 Analysis

3 Constructive Algorithm

3.1 Presentation

A heuristic algorithm is a type of algorithm that uses rules of thumb to try to find an approximate solution to a problem, rather than an exact one. Heuristic algorithms are often used when it is not possible to find an exact solution to a problem, or when an exact solution would be too time-consuming to compute. They are also used when an approximate solution is good enough, or when finding an exact solution is not the primary goal. Heuristic algorithms are commonly used in artificial intelligence, computer science, and other fields. They are often used to solve optimization problems, search problems, and other types of problems where an exact solution is not necessary or practical.

A constructive heuristic algorithm is a type of heuristic algorithm that is used to find a solution to a problem by building it incrementally. Constructive heuristics start with a partial solution and gradually add to it until a complete solution is found. They are commonly used to solve optimization problems, where the goal is to find the optimal solution, or the solution that is the best among all possible solutions.

Constructive heuristics can be contrasted with other types of heuristics, such as local search heuristics, which try to find a solution by making small changes to an existing solution, or random heuristics, which generate solutions randomly and then choose the best one. Constructive heuristics are often used when it is important to find a solution that is complete and comprehensive, rather than just a local improvement.

Examples of some famous problems that are solved using constructive heuristics are the flow shop scheduling, the vehicle routing problem and the open shop problem.

3.2 How it works

3.3 Pseudo code

3.4 Complexity

3.5 Instance

3.6 Experiments

3.7 Analysis

4 Local Search Algorithm

4.1 Presentation

4.2 How it works

4.3 Pseudo code

4.4 Complexity

4.5 Instance

4.6 Experiments

4.7 Analysis

5 Grasp Algorithm

5.1 Presentation

5.2 How it works

5.3 Pseudo code

5.4 Complexity

5.5 Instance

5.6 Experiments

5.7 Analysis

6 Conclusion

References

[1] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, September 1973. https://doi.org/10.1145/362342.362367.

[2] J.W. Moon and L. Moser. On cliques in graphs. *Israel J. Math.*, Match 1965. https://doi.org/10.1007/BF02760024.