



**UNIVERSITE MOHAMED BOUDIAF - M'SILA**  
**FACULTE DES MATHEMATIQUES ET**  
**DE L'INFORMATIQUE**



**DEPARTEMENT D'INFORMATIQUE**

**MEMOIRE de fin d'étude**

**Présenté pour l'obtention du diplôme de MASTER**

**Domaine : Mathématiques et Informatique**

**Filière : Informatique**

**Spécialité : Outils et méthode de l'informatique décisionnelle**

**Par : TAIBI Salah Eddine**

**SUJET**

**Experimental evaluation of a modified carousel algorithm  
For the minimum weight vertex cover problem**

**Soutenu publiquement le : 15 / 06 /2017 devant le jury composé de :**

**Nom et prénom Enseignant**

**Dr Salim Boumama .**

.....

.....

**Université de M'sila**

**Université de M'sila**

**Université de M'sila**

**Université de M'sila**

**Président**

**Rapporteur**

**Examineur**

**Examineur**

**Promotion : 2016 / 2017**

## **Acknowledgement**

*The first thanks is to Allah; then, to my supervisor*

*"Salim Bouamama"*

*who encourages me to do this research in English ;  
without him I wouldn't do it or develop  
my information as I am now, and not forgetting  
the great support of my family, friends, colleagues,  
and teachers . Needless to mention names  
because the paper is not enough to mention each  
of them all. Anyway, thank you so much.*

## Table of Contents

|                                                                       |    |
|-----------------------------------------------------------------------|----|
| Introduction.....                                                     | 05 |
| Chapter 1 : definition and fundamental concepts of graph theory ..... | 07 |
| 1.application of graph theory.....                                    | 08 |
| 1.1 What is graph .....                                               | 08 |
| 2.Definitions.....                                                    | 09 |
| 2.1 A simple graph.....                                               | 09 |
| 2.2 A directed graph.....                                             | 10 |
| 3. Types of graphs.....                                               | 10 |
| 4. Terminologies and basic notation.....                              | 12 |
| 4.1 Adjacency and incidence, neighborhood.....                        | 12 |
| 4.2 Degree and isolated vertex.....                                   | 13 |
| 4.3 Walks, trails, paths, circuits.....                               | 13 |
| 5. Graph representation.....                                          | 14 |
| 5.1 Adjacency list.....                                               | 15 |
| 5.2 Adjacency matrix.....                                             | 15 |
| 5.3 Comparison.....                                                   | 16 |
| 6. Some problem of graph theory.....                                  | 16 |
| 6.1 Travelling salesman problem.....                                  | 16 |
| 6.2 The minimum weight dominating set.....                            | 17 |
| Chapter 2 : overview of combinatorial optimization .....              | 18 |
| 1. What is Combinatorial Optimization?.....                           | 19 |
| 1.2 Why optimization.....                                             | 19 |
| 2. Solving optimization problems .....                                | 20 |
| 3. Formal definition of optimization problem.....                     | 21 |
| 3.1 Global optimum.....                                               | 21 |
| 3.2 Solution space.....                                               | 21 |
| 4. Classification of optimization problem.....                        | 22 |
| 5. Optimization vs decision.....                                      | 23 |
| 6. Algorithm and efficiency.....                                      | 23 |
| 6.1 Complexity classes P and NP.....                                  | 24 |
| 6.2 Reduction.....                                                    | 24 |
| 7. Optimization methods.....                                          | 25 |
| 7.1 Deterministic algorithms.....                                     | 25 |
| 7.2 Probabilistic algorithms.....                                     | 26 |

|                                                          |    |
|----------------------------------------------------------|----|
| chapter 3 : the minimum weight vertex cover .....        | 29 |
| 1.The vertex cover.....                                  | 30 |
| 1.2 Definition.....                                      | 30 |
| 1.3 Formal definition.....                               | 30 |
| 1.4 Complexity class.....                                | 30 |
| 1.5 Approximation algorithm for VC.....                  | 32 |
| 2. MWVC.....                                             | 32 |
| 2.1 Introduction.....                                    | 32 |
| 2.2 Formal definition of MWVC.....                       | 33 |
| 2.3 A LP Relaxation of Vertex Cover.....                 | 33 |
| 3. Related work.....                                     | 34 |
| 4. Real life -application of MWVC.....                   | 35 |
| chapter 4 : the carousel greedy heuristic CG .....       | 36 |
| 1.The greedy algorithm.....                              | 37 |
| 1.1 Introduction .....                                   | 37 |
| 1.2 Elements of the greedy strategy.....                 | 37 |
| 2. greedy heuristics for MWVC.....                       | 38 |
| 3.The Carousel greedy algorithm CG.....                  | 39 |
| 3.1 The basics steps of CG.....                          | 40 |
| 3.2 Pseudo-code description of CG for MWVC.....          | 41 |
| 3.3 Our carousel greedy algorithm.....                   | 42 |
| 3.4 Pseudo-code description of our a modified CG .....   | 43 |
| 3.5 future work of CG.....                               | 43 |
| Chapter 5 : Experimental results.....                    | 44 |
| 1.Experimental evaluations.....                          | 45 |
| 1.2 Problem instances.....                               | 45 |
| 2.Computational results of the greedy heuristics.....    | 45 |
| 3. Computational results of a modified CG algorithm..... | 50 |
| General conclusion.....                                  | 54 |
| Reference .....                                          | 55 |

## List of figures

|                                                                             |    |
|-----------------------------------------------------------------------------|----|
| Figure 1.1 Road map.....                                                    | 8  |
| Figure 1.2 Electrical network .....                                         | 8  |
| Figure 1.3 Graph .....                                                      | 9  |
| Figure 2. Simple graph .....                                                | 9  |
| Figure 3.1 Directed graph.....                                              | 10 |
| Figure 4.1 Complete graph.....                                              | 10 |
| Figure 4.2 Sub graph.....                                                   | 10 |
| Figure 4.3 Null graph .....                                                 | 11 |
| Figure 4.4 multigraph .....                                                 | 11 |
| Figure 4.5 A pseudo graph .....                                             | 11 |
| Figure 4.6 hypergraph.....                                                  | 11 |
| Figure 4.7 bipartite graph .....                                            | 12 |
| Figure 5 adjacent vertices .....                                            | 12 |
| Figure 6.1 simple graph .....                                               | 13 |
| Figure 6.2 multigraph.....                                                  | 13 |
| Figure 6.3 graph.....                                                       | 14 |
| Figure 7.1 Adjacency list .....                                             | 15 |
| Figure 7.2 adjacency matrix .....                                           | 15 |
| Figure 8.1 weighted graph to illustrate TSP .....                           | 16 |
| Figure 8.2 weighted graph to illustrate MWDS.....                           | 17 |
| Figure 9. global optimum and local optimum .....                            | 21 |
| Figure 10. Reductions to proof <i>NP</i> -completeness of some problem..... | 24 |
| Figure 11. Metaheuristic taxonomy.....                                      | 28 |
| Figure 12 optimization methods .....                                        | 28 |
| Figure 13 example of vertex cover .....                                     | 30 |
| Figure 14 Illustration of the construction in the proof 3-SAT.....          | 31 |
| Figure 15 The DNA double helix and SNP assembly problem.....                | 35 |
| Figure 16 Carousel greedy illustration.....                                 | 41 |

## List of table:

|                                                                |    |
|----------------------------------------------------------------|----|
| Table 1 : : classification of optimization problem .....       | 23 |
| Table 2 : experimentation on SPI instance .....                | 46 |
| Table 3 : experimentation on MPI instance.....                 | 47 |
| Table 4 : experimentation on LPI instance .....                | 48 |
| Table 5 : experimentation on MPI instance ( CG algorithm)..... | 50 |

# Introduction

Optimization is a scientific discipline that is concerned with obtaining the best result under given circumstances. Many challenging applications in business, economics, and engineering can be modeled as optimization problems. In fact, real-life problems solving are often complex and difficult to solve to which optimization plays a key role in finding feasible solutions to this problem.

Combinatorial optimization is a lively field of applied mathematics, combining techniques from combinatorics, linear programming, and the theory of algorithms, to solve optimization problems over discrete structures. A combinatorial optimization problem is defined over a set of instances each instance has a finite set of feasible solutions associated with it. Given an instance, the aim is to determine solutions that maximize or minimize a certain measure function.

The minimum weight vertex cover (MWVC) problem is a fundamental graph problem with many important real-life applications such as, for example, in wireless communication, circuit design and network flows. Given an undirected, vertex-weighted graph, the goal of the minimum weight vertex cover problem is to find a subset of the vertices of the graph such that the subset is a vertex cover and the sum of the weights of its vertices is minimal. This problem is known to be NP-hard and no efficient algorithm is known to solve it to optimality. Therefore, most existing techniques are based on heuristics for providing approximate solutions in a reasonable computation time [1].

In this work, we firstly provide comparison between different variations of a greedy heuristics for MWVC, besides, the greedy heuristics that have been used for the minimum dominating set [2] have been also used adopted for the MWVC problem, secondly the two best greedy heuristics obtained from the previous experiments have been investigated in the developments of **carousel greedy CG** algorithm the performance of our approach has been tested on a set of benchmark instances.

This thesis is structured as follows:

- In the first chapter, we discuss definition and concepts of graph theory pertinent to our study.
- In the second chapter, we talk about the basic concept of combinatorial optimization problems algorithms, including the complexity class of problems (Class P, class NP

and Class NP-complete), methods of resolution such as exact methods, approximate methods such as heuristics and metaheuristics.

- In the third chapter, we provide a deep description of the minimum weight vertex cover.
- In the fourth chapter, we explain the principles basics of greedy algorithm and carousel greedy algorithm CG also we discuss our algorithm
- In the last chapter, we report the results of an experimental performance evaluation of this approach on several well-known benchmark datasets.

# *Chapter 01*

## *definition and fundamental concepts of graph*

The field of mathematics plays vital role in various fields. One of the important areas in mathematics is **graph theory** which is used in structural models. This structural arrangement of various objects or technologies lead to new inventions and modifications in the existing environment for enhancement in those fields.[3]

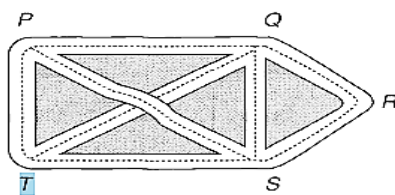
- **Application of graph:**

Graph theoretical concepts are widely used to study and model various applications, in different areas.[3]

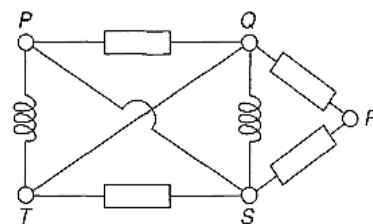
- **In chemistry:** They include, study of molecules, construction of bonds and the study of atoms.
- **In sociology:** graph theory is used to measure actors prestige or to explore diffusion mechanisms.
- **In biology:** Graph theory is used in biology and conservation efforts where a vertex represents regions where certain species exist and the edges represent migration path or movement between the regions. This information is important when looking at breeding patterns or tracking the spread of disease, parasites and to study the impact of migration that affect other species.
- **In operations research:** Graph theoretical concepts are widely used in this field. For example, the traveling salesman problem, the shortest spanning tree in a weighted graph, obtaining an optimal match of jobs and men and locating the shortest path between two vertices in a graph.
- **In computer science:** Especially in research areas of computer science such data mining, image segmentation, clustering, image capturing, networking etc., For example a data structure can be designed in the form of tree which in turn utilized vertices and edges. Similarly modeling of network topologies can be done using graph concepts.

### 1.1 what is a graph?

We begin by considering **Fig 1.1** and **Fig 1.2** which depict part of a road map and part of an Electrical network.

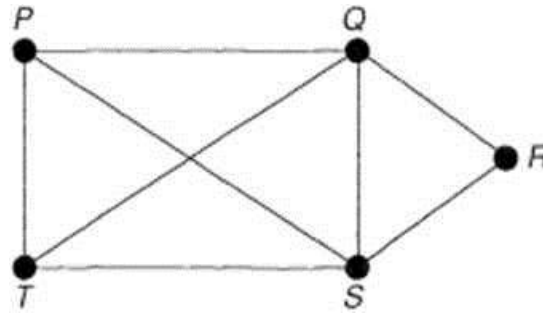


**Fig 1.1:** Road map



**Fig 1.2:** Electrical network

Either of these situations can be represented diagrammatically by means of points and lines, as in **Fig 1.3**. The points P, Q, P, S and T are called **vertices**, the lines are called **edges**, and the whole diagram is called a **graph**. [4]

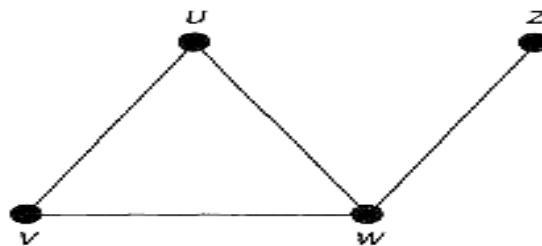


**Fig 1.3: graph**

- **Definitions:**

### 2.1 A simple graph:

A **simple graph** consists of a non-empty finite set  $V(G)$  of elements called vertices (or nodes), and a finite set  $E(G)$  of distinct unordered pairs of distinct elements  $V(G)$  called edges. We called edges. We call  $V(G)$  the vertex set and  $E(G)$  the edge set of  $G$ . An edge  $\{v, w\}$  is said to join the vertices  $v$  and  $w$ , and is usually abbreviated to  $vw$ . For example, **Fig 2** represent the simple graph whose vertex set  $V(G)$  is  $\{u, v, w, z\}$  and whose edge set  $E(G)$  consists of edges  $(uv)$ ,  $(uw)$ ,  $(vw)$  and  $(wz)$ . [4]



**Fig 2 :a simple graph**

## 2.2 A directed graph:

A directed graph or a digraph is a pair  $D = (V, E)$  where  $V$  is a finite set, and  $E$  is a finite family of ordered pairs of elements of  $V$ . the element of  $v$  are called the vertices or point, and the elements of  $V$  are called the vertices or points, and the elements of  $E$  are called the edges or arcs of  $D$ , the vertices  $v$  and  $w$  are called the tail and the head of the arc  $(v, w)$ , respectively. So the difference with undirected graphs is that orientations are given to the pairs.

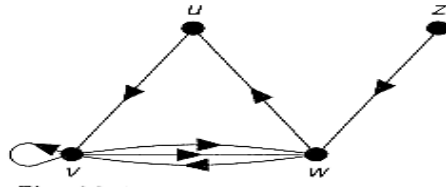


Fig 3: directed graph

### • Types of graphs:

- A simple graph that contains every possible edge between all the vertices is called a **complete graph**. a complete graph with  $n$  vertices is denoted as  $K_n$ . The first four complete graphs are given as examples[5]:

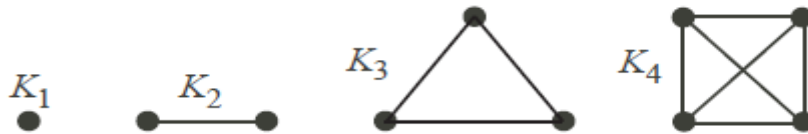


Fig 4.1:complete graph

- The graph  $G_1 = (V_1, E_1)$  is a **subgraph** of  $G_2 = (V_2, E_2)$  if
  - $V_1 \subseteq V_2$  And
  - $E_1$  is also an edge of  $G_2$

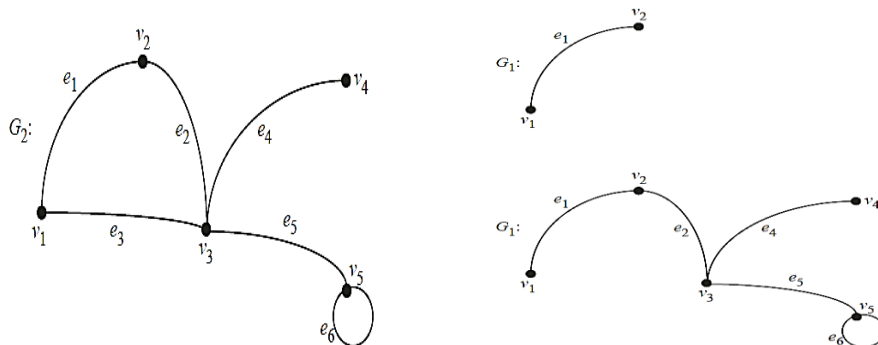


Fig 4.2: a sub graph

- In the definition of a graph  $G = (V, E)$  it is possible for the edge set  $E$  to be empty. Such a graph, without any edges, is called a **null graph**

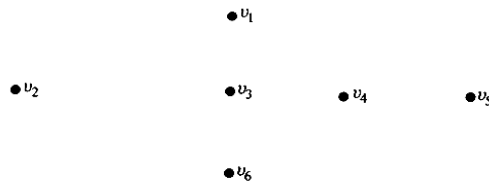


Fig 4.3: null graph

- If we allow repeated elements in our set of edges, technically replacing our set  $E$  with a multiset, we obtain a **multigraph**

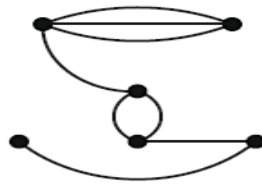


Fig 4.4 :multigraph

- By allowing edges to connect a vertex to itself ("loops"), we obtain a **pseudo graph**

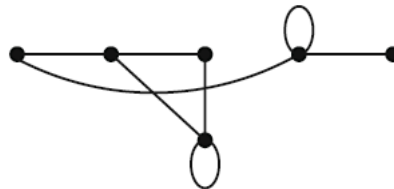


Fig 4.5: a pseudo graph

- Allowing our edges to be arbitrary subsets of vertices (rather than just pairs) gives us **hypergraphs**

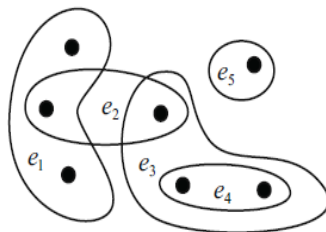


Fig 4.6: hypergraph

- A graph with only one vertex is **trivial**

●  $V_1$

- A simple graph  $G = (V, E)$  with vertex partition  $V = \{V_1, V_2\}$  is called a **bipartite graph** if every edge of  $E$  joins a vertex in  $V_1$  to a vertex in  $V_2$ . In general, a **Bipartite graph** has two sets of vertices, let us say,  $V_1$  and  $V_2$ , and if an edge is drawn, it should connect any vertex in set  $V_1$  to any vertex in set  $V_2$ .

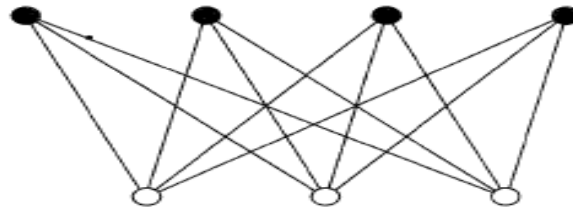


Fig 4.7 bipartite graph

#### 4. Terminologies and basic notation:

##### 4.1 Adjacency and incidence, neighborhood:

We say that two vertices  $v$  and  $w$  of a graph  $G$  are **adjacent** if there is an edge  $vw$  joining them, and the vertices  $v$  and  $w$  are then **incident** with such an edge. Similarly, two distinct edges  $e$  and  $f$  are **adjacent** if they have a vertex in common (see fig 5).



Fig 5 illustrate adjacency notation

**The neighborhood** (or open neighborhood) of a vertex  $v$ , denoted by  $N(v)$ , is the set of vertices adjacent to  $v$ :

$$N(v) = \{x \in V \mid vx \in E\}.$$

The closed neighborhood of a vertex  $v$  denoted by  $N[v]$ , is simply the set  $\{v\} \cup N(v)$ . Given a set  $S$  of vertices, we define the neighborhood of  $S$ , denoted by  $N(S)$ , to be the union of the neighborhoods of the vertices in  $S$ . Similarly, the closed neighborhood of  $S$ , denoted  $N[S]$ , is defined to be  $S \cup N(S)$ . [4]

## 4.2 Degree and isolated vertex:

The **degree** of a vertex  $v$  of  $G$  is the number of edges incident with  $v$ , and is written  $\text{Deg}(v)$  in calculating the degree of  $v$ , we usually make the convention that a loop at  $v$  contributes 2 (rather than 1) to the degree of  $v$ .

A vertex of degree 0 is an **isolated vertex** and a vertex of degree 1 is a **pendant vertex**. Thus, each of the two graphs in Fig 6.1. It has two end-vertices and three vertices of degree 2, while the graph in Fig 6.2. It has one end-vertex, one vertex of degree 3, one of degree 6 and one of degree 8 [4].

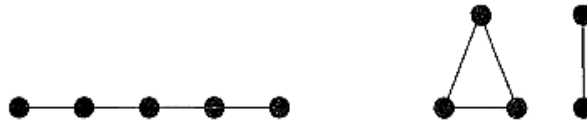


Fig 6.1 simple graph

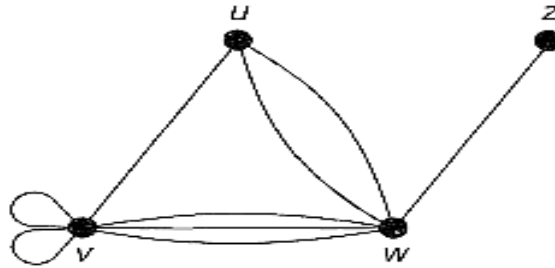


Fig 6.2 multiple graph

The **minimum degree** of the vertices in a graph  $G$  is denoted  $\delta(G)$ . ( $\delta(G) = 0$  if there is an isolated vertex in  $G$ ). similarly, we write  $\Delta(G)$  as the **maximum degree** of vertices in  $G$ .

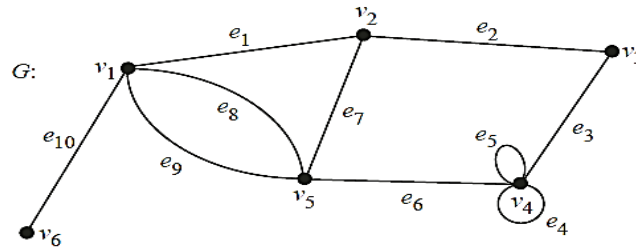
**Example.** (Continuing from the previous example fig 6.2).  $\delta(G) = 1$ .  $\Delta(G) = 8$ .

## 4.3 Walks, trails, paths, circuits:

- A **walk** in the graph  $G = (V, E)$  is a finite sequence of the form  $vi_0, e_{j_1}, vi_1, e_{j_2}, \dots, e_{j_k}, vi_k$ , which consists of alternating vertices and edges of  $G$ . The walk starts at a vertex. The vertices  $vi_{t-1}$  and  $vi_t$  are end vertices of  $e_{j_t}$  ( $t = 1, \dots, k$ ).  $vi_0$  is the **initial vertex** and  $vi_k$  is the **terminal vertex**.

$k$  is the **length** of the walk. A zero-length walk is just a single vertex  $v_{i_0}$ . It is allowed to visit a vertex or go through an edge more than once. A walk is **open** if  $v_{i_0} \neq v_{i_k}$ , otherwise it is closed.[5]

**Example:**



**Fig6.3: graph**

The walk  $v_2, e_7, v_5, e_8, v_1, e_8, v_5, e_6, v_4, e_5, v_4, e_5, v_4$  is opened.

. On the other hand, the walk  $v_4, e_5, v_4, e_3, v_3, e_2, v_2, e_7, v_5, e_6, v_4$  is closed.

- A walk is a **trail** if any edge is traversed at most once. Then, the number of times that the vertex pair  $u, v$  can appear as consecutive vertices in a trail is at most the number of parallel edges connecting  $u$  and  $v$ .

**Example.** (Continuing from the previous example) The walk in the graph

$v_1, e_8, v_5, e_9, v_1, e_1, v_2, e_7, v_5, e_6, v_4, e_5, v_4, e_4, v_4$  is a **trail**.

- A **trail** is a **path** if any vertex is visited at most once except possibly the initial and terminal vertices when they are the same. A closed path is a **circuit**. For simplicity, we will assume in the future that a circuit is not empty, i.e. its length  $\geq 1$ . We identify the paths and circuits with the subgraphs induced by their edges.[5]

**Example.** (Continuing from the previous example)

The walk  $v_2, e_7, v_5, e_6, v_4, e_3, v_3$  is a **path** and the walk  $v_2, e_7, v_5, e_6, v_4, e_3, v_3, e_2, v_2$  is a **circuit**.

## 5. Graph representation:

The two main graph representations we use when talking about graph problems are the **adjacency list** and the **adjacency matrix**. It's important to understand the tradeoffs between the two representations. Let  $G = (V, E)$  be our graph where  $V$  is the set of vertices and  $E$  is the set of edges where each edge is represented as a tuple of vertices [5]

## 5.1 Adjacency list

An adjacency list is a list of lists. Each list corresponds to a vertex  $u$  and contains a list of edges  $(u, v)$  that originate from  $u$ . Thus, an adjacency list takes up  $\Theta(V + E)$  space.

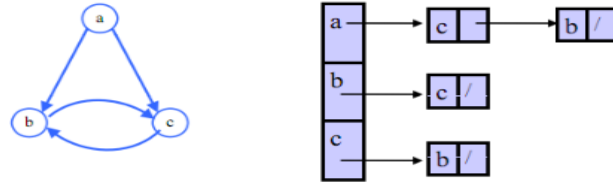


Fig 7.1: adjacency list

## 5.2 Adjacency matrix

An adjacency matrix is a  $|V| \times |V|$  matrix of bits where element  $(i, j)$  is 1 if and only if the edge  $(v_i, v_j)$  is in  $E$ . Thus, an adjacency matrix takes up  $\Theta(|V|^2)$  storage (note that the constant factor here is small since each entry in the matrix is just a bit).

➤ **Definition:**

Given an  $n$ -node graph  $G = (V, E)$  where  $V = \{v_1, v_2, v_3, v_4, \dots\}$  the adjacency matrix for  $G$  is the  $n \times n$  matrix  $A_G = \{a_{ij}\}$  where

$$a_{ij} = \begin{cases} 1, & \text{if } \exists (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$

If  $G$  is a weighted graph with edge weights given by  $w: E \rightarrow \mathbb{R}$  then the adjacency matrix for  $G$  is  $A_G = \{a_{ij}\}$  where

$$a_{ij} = \begin{cases} w(v_i, v_j), & \text{if } \exists (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

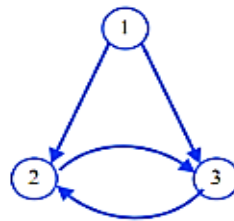


Fig 7.2 :adjacency matrix

### 5.3 Comparison

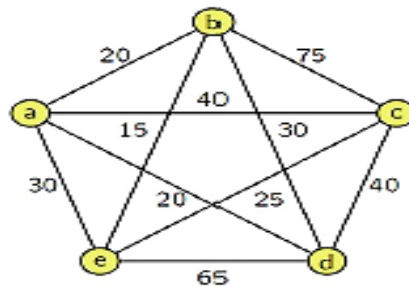
The worst-case storage of an adjacency list is when the graph is dense, i.e.  $E = \Theta(V^2)$ . This gives us the same space complexity as the adjacency matrix representation. The  $\Theta(V + E)$  space complexity for the general case is usually more desirable, however. Furthermore, adjacency lists give you the set of adjacent vertices to a given vertex quicker than an adjacency matrix  $O(\text{neighbors})$  for the former vs  $O(V)$  for the latter.[6]

## 6. Some problem of graph theory:

### 6.1 Travelling salesman problem:

Given a list of cities the salesman has to visit all cities (each exactly once) and return to the origin city. However his journey should be the shortest, or the cheapest, or the fastest. The distance (cost or time) of routing between each pair of cities is given. The problem can be formulated as the problem of finding optimal (shortest, cheapest or fastest) Hamiltonian cycle in a given graph [7]

consider a graph



**Fig 8.1: weighted graph to illustrate TSP**

- cycle a; b; c; d; e; a weight 230
- cycle a; b; e; c; d; a weight 110

Theoretically, TSP can be solved by setting  $(n-1)! / 2$  Hamiltonian cycles and choosing the cycle with the smallest total weight.

However, for  $n = 20$  cities we have to check  $19! / 2 = 60\,822\,550\,204\,416\,000$  cycles. This means that for a computer which is able to check 1 million cycles per second, checking all cycles will last approximately 1928.7 years!

Therefore, we can use:

- the direct method that generates the exact solution, but only for small  $n$ ,
- the approximate method, which generates solution close to the optimal, but working quickly.

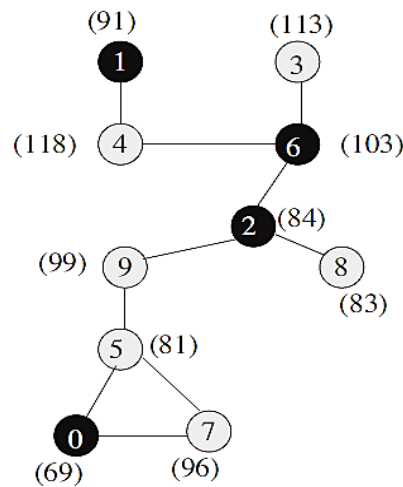
we will explain the approximate method and the direct method in the next chapter

## 6.2 The minimum weight dominating set:

Minimum weight dominating set (MWDS) finds many uses in solving problems as varied as clustering in wireless networks.

Given an undirected graph  $G = (V, E)$  where  $V$  is the set of vertices and  $E$  is the set of edges, a dominating set is a subset  $D \subseteq V$  such that every vertex  $v \in V$  is either a member of  $D$  or is adjacent to a vertex in  $D$ . A dominating set  $D'$  is called the minimum dominating set if its cardinality is minimum of all possible dominating sets [2].

**Example:**  $D' = \{1, 6, 2, 0\}$



**Fig 8.2: weighted graph to illustrate MWDS**

# *Chapter 2*

## *Overview of combinatorial optimization*

## 1. What is Combinatorial Optimization?

Combinatorial analysis is the mathematical study of the arrangement, grouping, ordering, or selection of discrete objects, usually finite in number. Traditionally, combinatorialists have been concerned with questions of existence or of enumeration. That is, does a particular type of arrangement exist? or, how many such arrangements are there? Quite recently, a new line of combinatorial investigation has gained increasing importance. The question asked is not “Does the arrangement exist?” or “How many arrangements are there? “, but rather, “What is a best arrangement?” The existence of a particular type of arrangement is usually not in question, and the number of such possible arrangements is irrelevant. All that matters is finding an optimal arrangement, whether it be one in a hundred or one in an effectively infinite number of possibilities. The new study of combinatorial optimization owes its existence to the advent of the modern digital computer. Most currently accepted methods of solution to combinatorial optimization problems would hardly have been taken seriously 25 years ago, for the simple reason that no one could have carried out the computations involved. Moreover, the existence of the digital computer has also created a multitude of technical problems of a combinatorial character. A large number of combinatorial optimization problems have been generated by research in computer design, the theory of computation, and by the application of computers to a myriad of numerical and nonnumerical problems which have required new methods, new approaches, and new mathematical insights. [8]

### 1.2 Why optimization ?

- Countless applications -science, engineering, business, economics
- Some form optimization is used in every company
- Every process can potentially be optimized
  - Minimize: time, cost, risk .....
  - Maximize: profit, quality, efficiency .....

A combinatorial optimization problem is either a maximization problem or a minimization Problem with an associated set of instances. Without loss of generality we will restrict ourselves to minimization problems, because every maximization problem can easily be converted into a Minimization problem.

## 2. Solving optimization problems:

Three steps to solve optimization problems[9] :

### ➤ Identify the problem

- Identify the decision problem
- Identify internal / external objectives
- Determine the input (parameters)
- Determine Constraints

### ➤ Modeling the problem

Model is simplification of reality. the quality of the solution depends on the quality of the model. the founded solution is for the abstract model and not the original .

There are two common approaches:

- **Simplified model, exact approach**

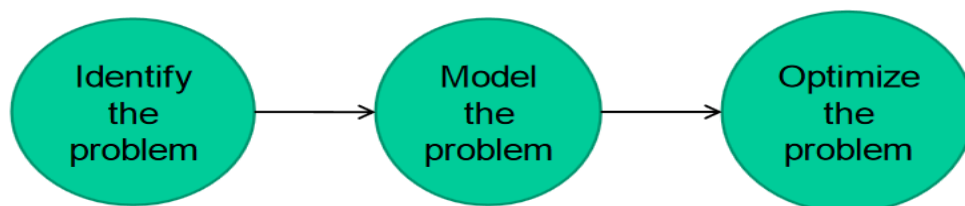


- **Precise model, approximate approach**



### ➤ Solving models

After we have defined a model of the original problem, the model can be solved by some kind of algorithm (usually an optimization algorithm). An algorithm is a procedure (a finite set of well-defined instructions) for accomplishing some task. an algorithm starts in an initial state and terminates in a defined end-state. The goal of an algorithm to find a solution (either specific values for the decision variables or one specific decision alternative) with minimal or maximal evaluation value.



### 3. Formal definition of optimization problem:

An instance of a combinatorial optimization problem is a pair  $(S, f)$  where [9] :

- $S$  is the set of feasible solution (solution space, search space)
- $f: S \rightarrow \mathfrak{R}$  is the objective function to optimize
- $S$  is defined by input parameters and constraints

In **discrete (or combinatorial) optimization** we concentrate on optimization problems, where for every instance  $I = (S', f)$  the set  $S$  of feasible solutions is **discrete**, i.e.,  $F$  is finite or countably infinite.

The main goal in solving an optimization problem is finding the **global optimum** (or one of them if there are multiple global optimal)

#### 3.1 Global optimum:

- A solution  $s^* \in S$  is a global optimum if
- ❖  $\forall s' \in S, f(s^*) \leq f(s')$  ( minimization problem)
- ❖  $\forall s' \in S, f(s^*) \geq f(s')$  ( maximization problem)

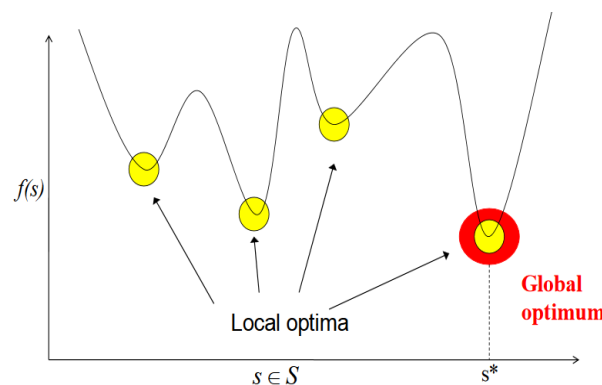


Fig 9 : global optimum and local optimum

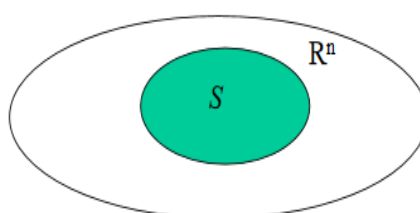
#### 3.2 Solution space:

$S$  is solution space for an optimization problem

- $S$  is empty that means no solution exists and the problem is too constrained
- $S$  is non-empty that means one or more (even finite) optimal solution can exist (with the same value of  $f$ )

$$S = \{ x \in \mathfrak{R}^n \text{ such that } x \text{ fulfils the constraint } \}$$

$$S \subseteq \mathfrak{R}^n$$



A combinatorial optimization problem can actually be attacked in three different versions which are:

**I. Search version:** given an instance  $(S, f)$ , find an optimal solution, that is an element  $S_{\text{opt}} \in S$ .

**II. Evaluation version:** given an instance  $(S, f)$ , find the optimal objective function value  $\text{opt } f(S)$

**III. Decision version:** given an instance  $(S, f)$  and a bound  $L$ , decide whether there is a feasible solution  $S' \in S$  with  $f(S') \leq L$ . [9]

#### 4. Classification of optimization problem

Most optimization models commonly used [9]:

- Mathematical programming
- Constraint programming

##### Mathematical program:

Minimise / maximise  $f(x_1, \dots, x_n) \rightarrow$  objective function  $(\mathbb{R}^n \rightarrow \mathbb{R})$  subject to :

➤ Inequality constraints

$$g_1(x_1, \dots, x_n) \geq b_1$$

.....

$$g_m(x_1, \dots, x_n) \geq b_m$$

➤ Equality constraints

$$h_1(x_1, \dots, x_m) = c_1$$

.....

$$h_m(x_1, \dots, x_m) = c_m$$

Optimization problem modeled using mathematical programming can be classified according to the nature of :

- The objective function  $f$  and constraint function  $g_i$  and  $h_i$
- The decision variable  $x$
- The type of input parameters

**Table 1: classification of optimization problem :**

| Classification                          | Refers to                                                               |
|-----------------------------------------|-------------------------------------------------------------------------|
| Linear vs nonlinear programming         | The type of objective function and / or Constraint                      |
| Discrete vs continuous programming      | The type of variable                                                    |
| Stochastic vs deterministic programming | The nature of parameters in the constraint<br>Or the objective function |
| Finite vs non finite programming        | The number of decision variable                                         |
| Constrained vs non-constraint           | Whether the constraints are defined or not                              |

## 5. Optimization vs decision:

In complexity theory, one usually considers decision problems instead of optimization problems.

Definition: A **decision problem** is given by a set of instances  $\Gamma$ . Each instance

$I \in \Gamma$  specifies

- A set  $F$  of feasible solution for  $I$
- A cost function  $c: F \rightarrow \mathbb{Z}$
- An integer  $K$

Given an instance  $I = (F, C, K)$  the goal is to decide whether there exists a feasible solution  $S \in F$  whose cost  $c(S)$  is at most  $K$ . If there is such a solution, we say that  $I$  is a “yes-instance”; otherwise,  $I$  is a “no-instance”. [10]

## 6. Algorithm and efficiency:

Intuitively, an **algorithm** for an optimization problem is a sequence of instructions specifying a computational procedure that solves every given instance  $I$  of the problem. Formally, the computational model underlying all our considerations is the one of a **Turing machine** efficiency refers to the overall running time of the algorithm. We actually do not care about the actual running time (in terms of minutes, seconds, etc.), but rather about the number of basic operations. there are different ways to represent the overall running time of an algorithm. The one that we will use here is the so-called worst-case running time or **computational complexity**. [10]

## 6.1 Complexity classes P and NP:

Intuitively, the complexity classes P and NP refer to decision problems that can be solved efficiently and those for which yes-instances can be verified efficiently, respectively. If we insisted on formal correctness here, we would define these classes in terms of a specific computer model called Turing machine. [10]

### a. Class P:

A decision problem belongs to the complexity class  $P$  if there exists an algorithm that for every instance  $I \in \Gamma$  determines in polynomial time whether  $I$  is a yes-instance or a no-instance.

### b. Class NP:

A decision problem belongs to the complexity class  $NP$  if every yes-instance  $I \in \Gamma$  admits a certificate whose validity can be verified in polynomial time. Note that the polynomial-time verifiability of  $S$  implies that the size of  $S$  must be polynomially bounded in  $|I|$  (because we need to look at  $S$  to verify its validity). That is the definition above also states that yes-instances of problems in NP have short, i.e., polynomially bounded, certificates.

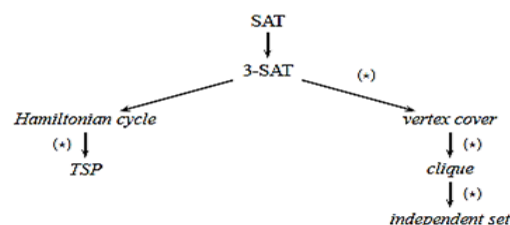
**Example:** the Hamiltonian cycle problem is in  $NP$ : A certificate for a yes-instance corresponds to a set of edges  $S \subseteq E$ . One can verify in  $O(n)$  time whether  $S$  constitutes a cycle in  $G$  that visits all vertices exactly once.

## 6.2 Reduction:

A problem  $P_1$  is polynomially reducible to problem  $P_2$ , if there is a process that takes an instance of  $P_1$  as an input, and outputs a corresponding instance of  $P_2$  in polynomial time.

- **Class NP-complete:**

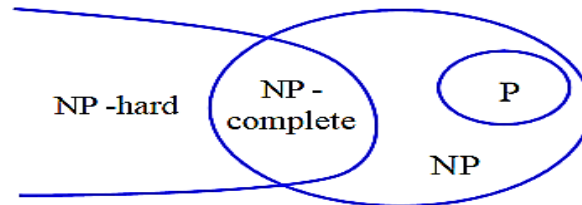
NP decision problem  $X$  is NP complete if all other problem in NP can be reduced to  $X$  in polynomial time



**Fig 10:** Reductions to proof NP-completeness of some problem

- **Class NP-hard:**

At least as hard as NP-complete problem, the problem can be any type: decision, search or optimization problems, all optimization problem whose associated decision problem are NP-complete are NP-hard .an algorithm for solving it can also solve NP-complete problem but an algorithm which solves an NP-complete problem does not necessarily solve it.



## 7. Optimization methods:

Generally, optimization algorithms can be divided in two basic classes: **deterministic** and **probabilistic** algorithms

### 7.1 Deterministic algorithms

Deterministic algorithms are most often used if a clear relation between the characteristics of the possible solutions and their utility for a given problem exists. Then, the search space can efficiently be explored using for example a divide and conquer scheme. If the relation between a solution candidate and its “fitness” are not so obvious or too complicated, or the dimensionality of the search space is very high, it becomes harder to solve a problem deterministically. Trying it would possible result in exhaustive enumeration of the search space, which is not feasible even for relatively small problems. [11]

- **Deterministic algorithm (Exact methods):**

- Simplex
- Branch and bound and A\*
- Dynamic programming
- Constraint programming

### 7.2 Probabilistic algorithms

Probabilistic Algorithms are those algorithms that model a problem or search a problem space using a probabilistic model of candidate solutions. Many Metaheuristics and Computational Intelligence algorithms may be considered probabilistic, although the difference with algorithms is the explicit (rather than implicit) use of the tools of probability in problem solving. They trade in guaranteed correctness of the solution for a shorter runtime. This does not mean that the results obtained using them are incorrect they may just not be the

global optima. On the other hand, a solution a little bit inferior to the best possible one is better than one which needs 10100 years to be found. [11]

### ➤ **Heuristics:**

Any approach that solves the problem without a formal guarantee on the quality of the solution can be considered as a heuristic for the problem. Some heuristics provide very good solutions in practice. An example of such an approach is *local search*: Start with an arbitrary solution and perform local improvement steps until no further improvement is possible. Moreover, heuristics are often practically appealing because they are simple and thus easy to implement.

Heuristics can be constrictive or improvement (or hybrid)

- **Constrictive** evaluate partial solution, terminate when a whole solution is constructed  
Ex: greedy algorithm
- **Improvement** iterative, evaluate complete solution  
Ex: metaheuristic

### ➤ **Approximation algorithm:**

Approximation algorithms are efficient algorithms that compute suboptimal solutions with a provable approximation guarantee. That is, here we insist on polynomial-time computation but relax the condition that the algorithm has to find an optimal solution by requiring that it computes a feasible solution that is “close” to optimal.

$\epsilon$ -approximation algorithm: its time complexity is polynomial and obtained solution  $S$  is

$$\text{If } \epsilon > 1, S \leq \epsilon \cdot S^*$$

$$\text{If } \epsilon < 1, S \geq \epsilon \cdot S^*$$

where  $S^*$  is global optimum (minimization problem) and  $\epsilon$  can be constant or a function of the input size.

### ➤ **Metaheuristic algorithm:**

A metaheuristic is formally defined as an iterative generation process with guided a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solution [ 11]

other definition of metaheuristic:

A top level general strategy which guides other heuristic to search for feasible solution in domains where the task is hard.

meta (from Greek: heuriskein = to find to discover) heuristic (from Greek: beyond the sense of height-level). [ref 11]

The main aim is to ensure a balance between:

- **Exploration** of search space
- **Exploitation** of the information gathered during the previous search

I. Metaheuristics which use a candidate solution:

- ✓ Local search LS
- ✓ Simulated annealing SA
- ✓ Tabu search TS
- ✓ Greedy randomized adaptive search GRASP

II. Metaheuristics which use a population solution:

- Evolutionary algorithms :
  - ✓ Evolution strategies ES
  - ✓ Evolution programming EP
  - ✓ Genetic algorithm GA
  - ✓ Genetic programming GP
- Swarm intelligence algorithm:
  - ✓ Particle swarm optimization PSO
  - ✓ Ant colony optimization ACO
  - ✓ Artificial bee colony ABC

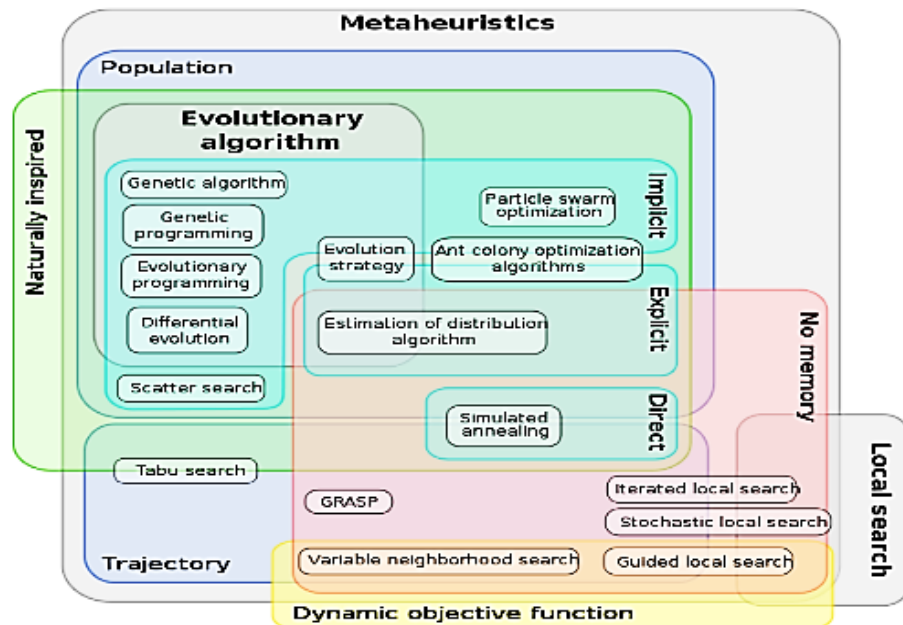


Fig 11: Metaheuristic taxonomy

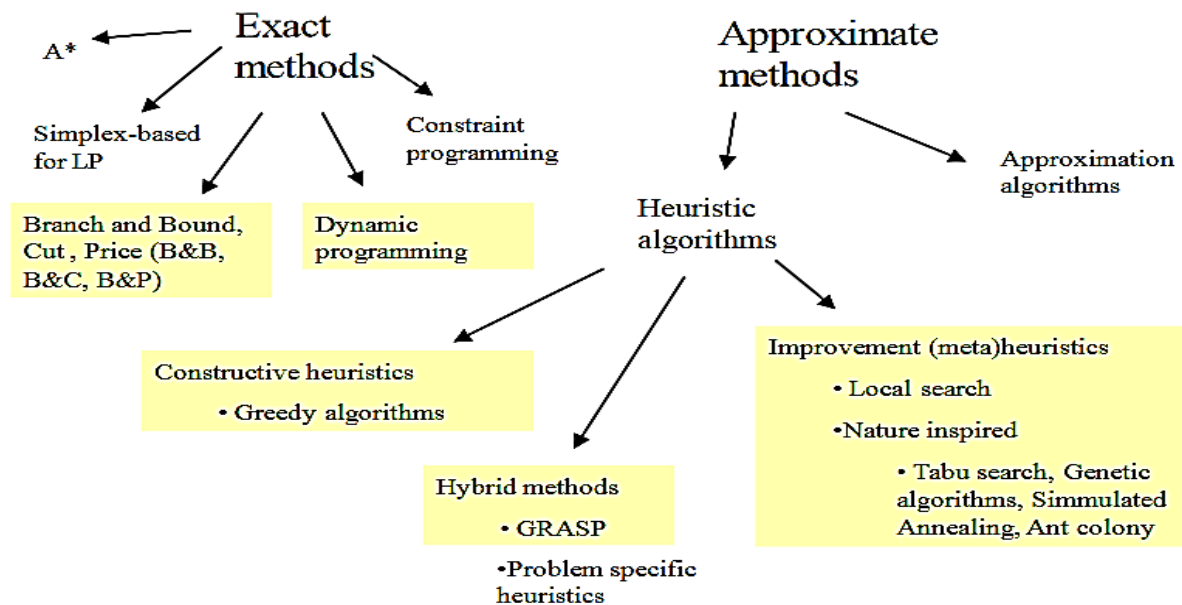


Fig 12 Optimization methods

## *Chapter 03*

*The minimum weight  
vertex cover problem*

## 1. The vertex cover:

The vertex cover problem is a fundamental graph problem with many important real-life applications.

### 1.2 Definition:

A vertex cover of a graph is a set of vertices that "covers" the edges of the graph. That is, every edge has at least one endpoint in the covering set, or, equivalently, for every vertex, it is either in the set or adjacent to some vertex which is in the set.

### 1.3 Formal definition:

Given an undirected graph  $G = (V, E)$  a vertex cover of  $G$  is a subset of the vertices  $V' \subseteq V$  such that for any edge  $(u, v) \in E$ , then  $u \in V'$  or  $v \in V'$  or both [12].

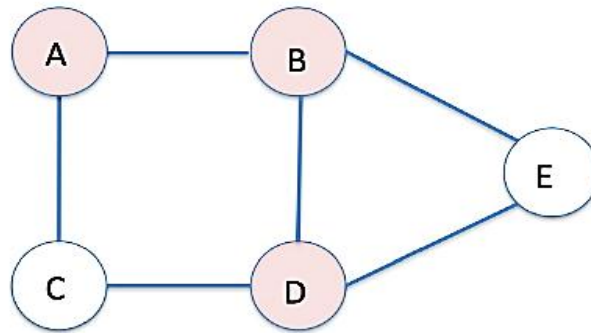


Fig 13:example of vertex cover

### 1.4 Complexity class:

**Proof.** We first argue that **vertex cover** is in NP. A certificate of a yes-instance is a subset  $V' \subseteq V$  of vertices with  $|V'| \leq K$  that forms a vertex cover of  $G = (V, E)$ . This can be verified in time at most  $O(n+m)$  by checking whether each edge  $(u, v) \in E$  has at least one of its incident vertices in vertices in  $V'$ .

In order to prove that vertex cover is NP-complete, we will show that  $3\text{-SAT} \leq \text{Vertex cover}$ . Note that this is sufficient because 3-SAT is NP-complete.

We transform an instance of 3-SAT to an instance of vertex cover as follows: Consider a Boolean formula  $F$  in CNF with each clause having three literals. Let  $n$  and  $m$  denote the number of variables and clauses of  $F$ , respectively. We create a *variable-gadget* for each variable  $x$  consisting of two vertices  $x$  and  $\neg x$  that are connected by an edge. Moreover, we create a clause-gadget for each clause  $C = (l_1 \vee l_2 \vee l_3)$  consisting of three vertices  $l_1, l_2, l_3$  that

are connected by a triangle. Finally, we connect each vertex representing a literal in a clause-gadget to the corresponding vertex representing the same literal in the variable gadget.

Let  $G = (V, E)$  be the resulting graph; see Fig 14 for an example. Note that this transformation can be done in polynomial time.

We show that  $F$  is satisfiable if and only if  $G$  has a vertex cover of size at most  $K = n+2m$ .

First note that every assignment satisfying  $F$  can be turned into a vertex cover of size  $K$ :

For each variable-gadget we pick the vertex that corresponds to the literal which is true.

This covers all edges in the variable-gadgets and their respective connections to the gadgets. For each clause-gadget we choose two additional vertices so as to ensure that all remaining edges are covered. The resulting vertex cover has size  $K = n+2m$  as claimed. Next suppose that we are given a vertex cover  $V'$  of  $G$  of size at most  $K$ . Note that every vertex cover has to pick at least one vertex for every variable-gadget and two vertices for each clause-gadget just to cover all edges inside these gadgets. Thus,  $V'$  contains exactly  $K$  vertices. The vertices in  $V'$  now naturally induce an assignment as described above that satisfies  $F$ . We conclude that yes-instances correspond under the above reduction, which completes the proof. so the decision version of this problem is NP-complete. [10]

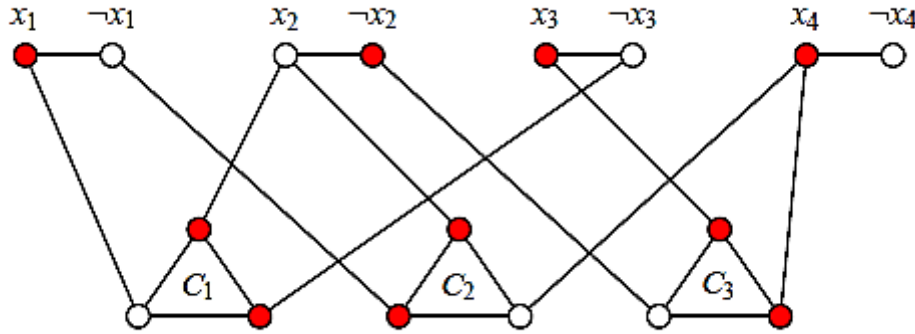


Fig 14: Illustration of the construction in the proof

### 1.5 Approximation algorithm for vertex cover problem

An algorithm for a problem of size  $n$  has an approximation ratio  $\Gamma(n)$  if for any input, algorithm produces a solution of cost  $C$  such that:  $\text{Max} \left( \frac{C}{C_{opt}}, \frac{C_{opt}}{C} \right) \leq \Gamma(n)$  such an algorithm is called a  $\Gamma(n)$ -approximation algorithm.

An approximation scheme that takes as input  $\epsilon > 0$  and produces a solution such that  $C = (1+\epsilon) C_{opt}$  for any fixed  $\epsilon$ , is a  $(1+\epsilon)$ -approximation algorithm.

A Polynomial Time Approximation Scheme (PTAS) is an approximation algorithm that runs in time polynomial in the size of the input  $n$ . [13].

#### Approx.-vertex-cover

1.  $C \leftarrow \emptyset$
2.  $E' \leftarrow E$
3. **While**  $E \neq \emptyset$
4.   **pick**  $(u, v) \in E$  arbitrarily
5.    $C \leftarrow C \cup \{u\} \cup \{v\}$
6.   **Delete** from  $E'$  all edge incident on  $u$  or  $v$
7. **Return**  $C$ .

Approximation-vertex-cover is a 2-approximation algorithm

**Proof:** let  $A$  denote the edges that are picked optimal cover  $C_{opt}$  must include at least one endpoint of each edge in  $A$ . no two edges in  $A$  share an endpoint.

$|A|$  is lower bound for  $|C_{opt}|$ ,  $|C_{opt}| \geq |A|$  number of vertices in  $C = 2|A|$ ,  $C \leq 2|C_{opt}|$

## 2. The minimum weight vertex cover problem MWVC:

### 2.1 Introduction

The minimum weight vertex cover (MWVC) problem is a fundamental graph problem with many important real-life applications such as, for example, in wireless communication, circuit design and network flows. Given an undirected graph  $G = (V, E)$  with weights on its vertices, the goal of the minimum weight vertex cover problem is to find among all subsets  $S \subset V$  that are vertex covers a subset  $S^*$  for which the sum of the weights of the vertices is minimal. The decision version of the problem is NP-complete [1].

## 2.2 Formal definition of MWVC

A problem instances  $(G, \omega)$  of the MWVC problem is a tuple that consists of an undirected graph  $G(V, E)$ , where  $V$  is the set of vertex and  $E$  is the set of edges, and a function  $\omega: V \rightarrow \mathbb{R}^+$  that associates a positive weight value  $\omega(v)$  to each vertex  $v \in V$ .

The MWVC problem can be formally defined as follows [1]:

$$\text{Minimize } \omega(S) = \sum \omega(v)$$

$$\text{Subject } \forall (v_i, v_j) \in E : (v_i \in S) \text{ or } (v_j \in S) \quad S \subseteq V$$

In the above definition, a candidate solution to the problem, denoted by  $S$ , is a subset of  $V$ . a candidate solution  $S$  is a valid solution, a so-called vertex cover, if each edge in  $G$  has at least one endpoint in  $S$ . The objective function value  $\omega(S)$  of a candidate solution  $S$  is defined as the sum of the weights of the vertices in  $S$ . The objective of the MWVC problem is to find a valid candidate solution that minimizes the objective function.

we denote by  $n = |V| = \{v_1, v_2, \dots, v_n\}$  the number of vertices and by  $m = |E|$  the number of edges in  $G$ . Moreover, we assume that  $G$  is given by means of an adjacency matrix  $A$ .

$$A = \{a_{ij} \mid i = 1, \dots, n; j = 1, \dots, n\} \text{ where}$$

$$A_{ij} = \begin{cases} 1, & \text{if } \exists (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$

Finally, we denote by  $d(v)$  the degree of a vertex  $v \in V$ , that is, the number of edges incident to this vertex, which is defined as

$$d(v) = \sum_{j=1}^n a_{ij} = \sum_{j=1}^n a_{ji}$$

## 2.3 A Linear Programming Relaxation of Vertex Cover:

we can formulate the minimum vertex cover problem as an ILP by using a variable  $x_v$  for each vertex  $v$ , taking the values 0 or 1, with the interpretation that  $x_v = 0$  means that  $v \notin S$  and  $x_v = 1$  means that  $v \in S$ . The cost of the solution, which we want to minimize, is  $\sum_{v \in V} x_v c(v)$  and we want  $x_u + x_v \geq 1$  for each edge  $(u, v)$  this gives the ILP [12]

$$\text{minimize } \sum_{v \in V} x_v c(v)$$

subject to

$$x_u + x_v \geq 1 \quad \forall (u, v) \in E$$

$$x_v \leq 1 \quad \forall v \in V$$

$$x_v \in \mathbb{N} \quad \forall v \in V$$

next, we relax the ILP to linear program.

$$\begin{aligned}
 & \text{minimize } \sum_{v \in V} x_v c(v) \\
 & \text{subject to} \\
 & \quad x_u + x_v \geq 1 \quad \forall (u, v) \in E \\
 & \quad x_v \leq 1 \quad \forall v \in V \\
 & \quad x_u \geq 0 \quad \forall v \in V
 \end{aligned}$$

### 3. Related work:

Due to its computational complexity as a NP-complete problem, most developed techniques are based on heuristics for providing approximate solutions in reasonable computation time.

The first one was introduced in [ 14]. It starts with an empty partial solution and adds one vertex at a time. More specifically, at each step it chooses the vertex with the minimum ratio between vertex weight and current degree. Note that the current degree is only determined by edges that are not yet covered.

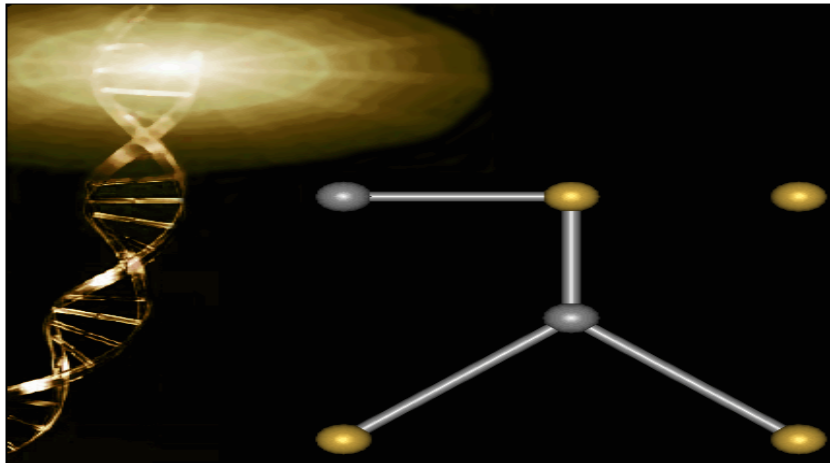
We mention Population-based search approaches. These have shown to be effective for solving a multitude of combinatorial optimization problems. Their advantage can be identified as their ability to find areas of the space containing high quality solutions. This work is done by [salim bouamama], he proposes a simple and efficient population-based iterated greedy algorithm(PBIG) for tackling the minimum weight vertex cover problem. At each iteration, a population of solutions is established and refined using a fast-randomized iterated greedy heuristic based on successive phases of destruction and reconstruction. [ 1]

On the other hand [ SHYONG] introduces a meta-heuristic based upon the Ant Colony Optimization (ACO) approach, to find approximate solutions to the minimum weight vertex cover problem. In the literature, the ACO approach has been successfully applied to several well-known combinatorial optimization problems whose solutions might be in the form of paths on the associated graphs. A solution to the minimum weight vertex cover problem however needs not to constitute a path. The ACO algorithm proposed incorporates several new features so as to select vertices out of the vertex set whereas the total weight can be minimized as much as possible. Computational experiments are designed and conducted to study the performance of our proposed approach. Numerical results evince that the ACO algorithm demonstrates significant effectiveness and robustness in solving the minimum weight vertex cover. [15]

The new greedy algorithm is proposed by [ Golden, Cerulli, Cerrone] in April 2017. they introduce carousel greedy, an enhanced greedy algorithm which seeks to overcome the traditional weaknesses of greedy approaches. They have applied carousel greedy to a variety of well-known problems in combinatorial optimization such as the minimum label spanning tree problem, the minimum vertex cover problem, the maximum independent set problem, and the minimum weight vertex cover problem. In all cases, the results are very promising. Since carousel greedy is very fast, it can be used to solve very large problems. In addition, it can be combined with other approaches to create a powerful, new metaheuristic. [16]

#### 4. Real life -application of MWVC:

In **computational biochemistry**, there are many situations where we wish to resolve conflicts between sequences in a sample by excluding some of the sequences. Of course, exactly what constitutes a conflict must be precisely defined in the biochemical context. We define a conflict graph where the vertices represent the sequences in the sample and there is an edge between two vertices if and only if there is a conflict between the corresponding sequences. The aim is to remove the fewest possible sequences that will eliminate all conflicts. the aim is to find a minimum vertex cover in the conflict graph  $G$ . We look at a specific example of the SNP assembly problem and show how to solve this problem using the vertex cover algorithm [ 17]



**Fig 15 :** The DNA double helix and SNP assembly problem

# *Chapter 4*

*the carousel greedy  
heuristic*

### 1. The greedy algorithm :

#### 1.1 Introduction :

Algorithms for optimization problems typically go through a sequence of steps, with a set of choices at each step. For many optimization problems, using dynamic programming to determine the best choices is overkill; simpler, more efficient algorithms will do. A **greedy algorithm** always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.

Greedy algorithms do not always yield optimal solutions, but for many problems they do.

[18]

#### 1.2 Elements of the greedy strategy:

A greedy algorithm obtains an optimal solution to a problem by making a sequence of choices. For each decision point in the algorithm, the choice that seems best at the moment is chosen. This heuristic strategy does not always produce an optimal solution, we design greedy algorithms according to the following sequence of steps:

- A. Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.
- B. Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe.
- C. Demonstrate that, having made the greedy choice, what remains is a subproblem with the property that if we combine an optimal solution to the subproblem with the greedy choice we have made, we arrive at an optimal solution to the original problem.

[18]

#### ➤ The algorithm

##### Greedy Template

1. R is the set of all requests
2. S is empty (\* solution of the problem \*)
3. **While** R is not empty **do**
4.   Choose  $i \in R$
5.   Add i to S
6.   Remove from R all requests that overlap with i
7. **Return** S

## 2. greedy heuristics for MWVC:

We experiment five different variations of a greedy heuristic [2] to determine the minimum weight vertex cover **MWVC**. the aim of this section is to compare the performance of the five different heuristics with each other. and we chose the two best heuristics. We use it in the next section.

In the **first heuristic**, we use an algorithm based on the standard greedy algorithm for the weighted set-cover problem [14]. The vertex cover set being constructed,  $V'$  is initialized to a NULL set in the beginning. The degree of a node,  $d(u)$ , is the number of neighbors of a given node  $u$ . For computing the minimum weight vertex cover, we divide the degree of the node,  $d(u)$ , by the weight of the node  $w(u)$  and add the node with the maximum ratio to the vertex cover set  $V'$ . If  $S$  represents  $V - V'$  where  $V'$  is the set of vertex cover nodes then the next node to include in  $V'$  is as follows:

$$V \leftarrow \arg \max_{u \in S} \frac{d(u)}{w(u)}$$

**Algorithm :**

**input:**  $G = (V, E)$

**output:** vertex cover set of  $G$  and  $\omega(S)$

1.  $S \leftarrow \emptyset$
2. **While**  $E \neq \emptyset$
3. **Pick a** vertex  $v \in V$ .  $V \leftarrow \arg \max_{v \in S} \frac{d(v)}{w(v)}$
4.  $S \leftarrow S \cup \{v\}$
5.  $E \leftarrow E \setminus \{e \in E: v \in V\}$
6. **Return**  $S$ .

In the **second heuristic**, we divide the degree to the power of two of the node  $d(u)^2$ , by the weight of the node  $w(u)$  and add the node with the maxi-mum ratio to the vertex cover set  $V'$ . If  $S$  represents  $V - V'$  where  $V'$  is the set of vertex cover nodes. then the next node to include in  $V'$  is as follows:

$$V \leftarrow \arg \max_{u \in S} \frac{d(u)^2}{w(u)}$$

In the **third heuristic**, we divide the sum of the weights of the neighbors of a node by the weight of the node. The node with the maximum ratio is taken as the node to be included in the vertex cover set. The weight of the neighbors of a node  $u$  is represented as:

$$W(u) = \sum_{s \in N(u)} w(s)$$

$$V \leftarrow \arg \max_{u \in S} \frac{W(u)}{w(u)}$$

In the **fourth heuristic**, we calculate the difference of the weight of the neighbors of a node and its own weight. The node with the maximum difference is chosen as the node to be included in the vertex cover set.  $V \leftarrow \arg \max_{u \in S} (W(u) - w(u))$

In the **fifth heuristic**, we divide the product of the degree of a node and the sum of the weights of the neighbors of a node by the weight of the node. The node with the maximum ratio is taken as the node to be included in the vertex cover set.

$$V \leftarrow \arg \max_{u \in S} \frac{W(u) \times d(u)}{w(u)}$$

### 3. The Carousel greedy algorithm:

Greedy algorithms have been developed for a large number of problems in combinatorial optimization. For many of these greedy algorithms, elegant worst-case analysis results have been obtained. These greedy algorithms are typically very easy to describe and code and they have very fast running times. On the other hand, the accuracy of greedy algorithms is often unacceptable. When instance size is large, exact solution approaches are generally not practical. However, a wide variety of metaheuristics (e.g., tabu search, genetic algorithms, variable neighborhood search, etc.) have been successfully applied to solve combinatorial optimization problems to within 5% (or better) of optimality. In contrast to greedy algorithms, metaheuristics can be complex and difficult to code. Running times can grow quickly with instance size. One key question that arises is as follows: Is there a middle ground between greedy algorithms and metaheuristics? In other words, can we identify a class of heuristics that performs nearly as well as metaheuristics with respect to accuracy, but has running times (and the simplicity) that are similar to what we see with greedy algorithms?

The carousel greedy algorithm is an enhanced greedy algorithm which, in comparison to a greedy algorithm, examines a more expansive space of possible solutions with a small and predictable increase in computational effort

An outline of the carousel greedy algorithm (CG) is as follows:

- I. Create a partial solution using a greedy algorithm;
- II. Use the same greedy algorithm to modify the partial solution in a deterministic way;
- III. Apply the greedy algorithm to produce a complete, feasible solution

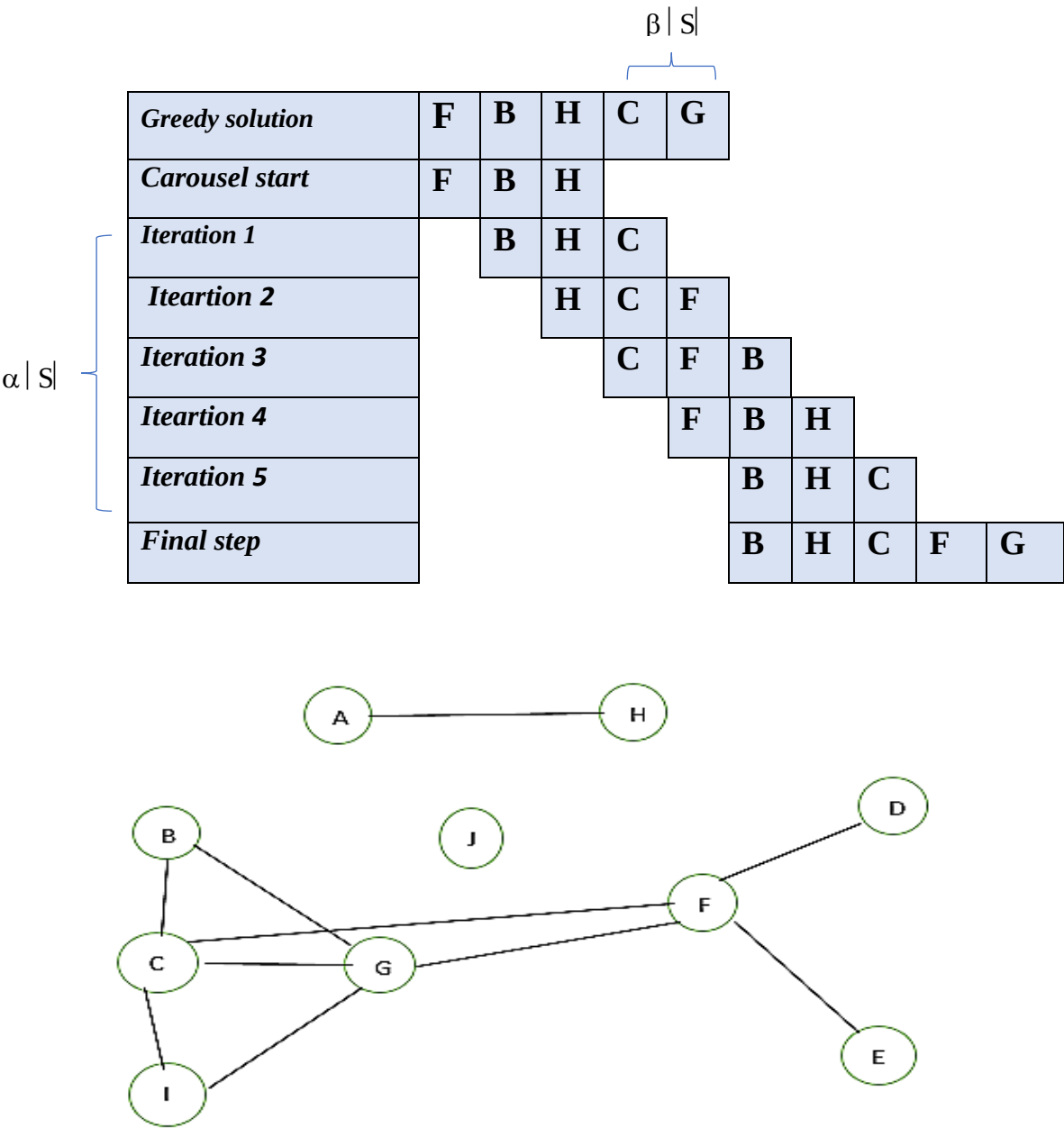
Unlike a typical metaheuristic, CG does not progress from one feasible solution to another, hopefully better, feasible solution. Only one feasible solution, at the end of the algorithm, is identified. In fact, CG does not calculate an objective function until the final step. One can consider the greedy and the CG algorithms to be constructive metaheuristics in contrast to improving metaheuristics. [16]

### 3.1 The basics steps of carousel algorithm:

1. Apply greedy algorithm to obtain a feasible solution.
2. Destruction phase: Remove some elements from the current solution, leaving a partial solution.
3. Construction phase: Apply the greedy algorithm to the partial solution to obtain another feasible solution.
4. Repeat Steps 2 & 3 until a stopping condition is satisfied. [16]

We will now introduce the carousel greedy algorithm using the schematic displayed in Figure 16. CG requires that two parameters,  $\alpha$  and  $\beta$ , be specified, where  $\alpha$  is an integer and  $\beta$  is a percentage. We can think of the letters as vertices of the graph  $G$  in figure 16. In row 1, we begin with an execution of the greedy algorithm (the first heuristic  $v \leftarrow \arg \max_{u \in S} \frac{d(u)}{w(u)}$ ). the solution is  $F B H C G \mid S \mid = 5$  vertices, in row 2, we drop the last  $\beta = 40\%$  of the  $l$  selected in the first row to leave us with  $F, B$ , and  $H$ . ( $G$  is not cover). The idea is to postpone the last 40% or so of selections up to the very end. Until then, we will maintain a set of  $(1 - \beta)|S|$  vertices from row to row. Essentially, we are prolonging phase II. Before moving on to row 3, we drop the first vertex  $F$  in row 2. Next given the remaining vertices  $B H$  ( $G$  is not cover). we apply the greedy algorithm to select third vertex  $C$ . yielding  $B, H$  and  $C$  in row 3, we drop  $B$  from row 3 apply the greedy heuristic to select  $F$  and obtain  $H, C$  and  $F$  for row 4. We continue in this manner until we reach row 7. At this point, we have vertices  $B, H$  and  $C$  and we have completed 5 iterations since the start of carousel greedy. With  $\alpha = 1$ , in this example, we stop what is essentially an extended Phase II (note that  $\alpha$  controls the degree to which

Phase II is extended) after  $\alpha|S| = 5$  iterations and proceed to what is essentially a short Phase III. In row 8, we start with the partial solution from row 7, apply the greedy algorithm to obtain a feasible solution, and then stop.



**Fig16** : Carousel greedy illustration for  $|S| = 5$ ,  $\alpha = 1$ ,  $\beta = 40\%$

### 3.2 Pseudo-code description of CG for MWVC [16] :

**Input:** I, G (I is a generic input instance, G is greedy algorithm)

1. Let  $S \leftarrow$  the solution produced by G  
(order by sequence of selection, the last selected elements are in head)
2.  $R \leftarrow$  the partial solution produced by removing from head of S,  $\beta|S|$  elements
3. Let  $\gamma \leftarrow$  the objective value associated with partial solution R
4. **For**  $\alpha|S|$  **iteration**
5. remove from tail of R one or more elements until the objective function value associated with the partial solution R is  $\leq \gamma$
6. using G add an element to head of R
7. **end for**
8. using G, add elements to R to obtain a feasible solution
9. return R.

### 3.3 A modified carousel greedy algorithm:

we modify this version of CG, we chose two best greedy heuristics and we apply in CG. let G1, G2 the two best greedy heuristics, the determinism rate  $\Psi$  is used to decide which one of two greedy is chosen. this done by firstly generating a random value  $\Psi$  uniformly distributed over  $[0, 1]$ . If  $\Psi \leq 0.5$  G1 is chosen, otherwise G2 is chosen, note  $\Psi$  is generated on every iteration in construction phase.

### 3.4 Pseudo-code description of a modified CG :

**Input:**  $I, G1, G2, \Psi$  ( $I$  is a generic input instance,  $G1, G2$  are greedy algorithms and parameter  $\Psi$ )

1.  $\Psi \leftarrow$  random number uniformly distributed over  $[0.1]$
2. **If**  $\Psi \leq 0.5$  **then**  $S \leftarrow$  the solution produced by  $G1$
3. **Else**  $S \leftarrow$  the solution produced by  $G2$   
(order by sequence of selection, the last selected elements are in head)
4.  $R \leftarrow$  the partial solution produced by removing from head of  $S$   
 $\beta |S|$  elements
5. Let  $\gamma \leftarrow$  the objective value associated with partial solution  $R$
6. **For**  $\alpha |S|$  **iteration**
7. remove from tail of  $R$  one or more elements until the objective function value associated with the partial solution  $R$  is  $\leq \gamma$
8.  $\Psi \leftarrow$  random number uniformly distributed over  $[0.1]$
9. **If**  $\Psi \leq 0.5$  then using  $G1$  to add an element to head  $R$
10. **Else** using  $G2$  to add an element to head  $R$
11. **end for**
12.  $\Psi \leftarrow$  random number uniformly distributed over  $[0.1]$
13. **If**  $\Psi \leq 0.5$  then using  $G1$  to add an element to head  $R$
14. **Else** using  $G2$  to add an element to head of  $R$
15. return  $R$

### 3.5 future work of CG:

In future work, there are a number of interesting directions to explore. First, we expect to conduct more comprehensive computational studies of CG stepwise regression and Hybrid CG. Second, as pointed out by Glover et al. and others, recent advances in technology have led to new applications involving very large to huge problem instances in which one-pass heuristics are required. Since CG is so efficient, we plan to test it on some very large problem. In future work, we might try to apply CG to a different class of optimization problems, like the traveling salesman problem [16].

# *Chapter 05*

## *Experimental results*

## 1. Experimental evaluations :

The proposed algorithms were implemented in C++ using visual studio2017 as IDE, and experimentally evaluated on pc with Intel® core™ i3-4005U CPU @1.70 GHz and 4 GB RAM.

### 1.2 Problem instances:

The algorithms were tested on a set of benchmark instances [15]. each instance consists of an undirected, vertex-weighted graph with  $n$  vertices and  $m$  edges. These instances are grouped with respect to their number of vertices  $n$  into three different classes[1]:

1. **Class SPI:** the class of small problem instances (SPI) contains 400 instances. The number of nodes ( $n$ ) of this instances from this class takes values from {10, 15, 20, 25}
2. **Class MPI:** the class of moderate-size problem instance (MPI)-consists of 710 instances where  $n$  takes values from {50,100,150,200,250,300}
3. **Class LPI:** the class of large problem instances (LPI) consists of 15 instances with  $n$  from {500 ,800,1000}

In each class – and for each  $n$  – a whole range of instances exists, including rather sparse graphs as well as rather dense graphs. The instances of the first two classes share the following characteristics: (i) 10 problem instances were randomly generated per combination of  $n$  and  $m$  and results are presented as an average over the objective function values obtained for the 10 instances. (ii) The weight  $\omega(v)$  of each vertex  $v \in V$  is randomly drawn from a uniform distribution, either from the interval [20, 120]. class LPI only consists of one problem instance per combination of  $n$  and  $m$  and the average quality over 10 independent runs are generally reported as quality measure of an algorithm.

## 2. Computational results of the greedy heuristics:

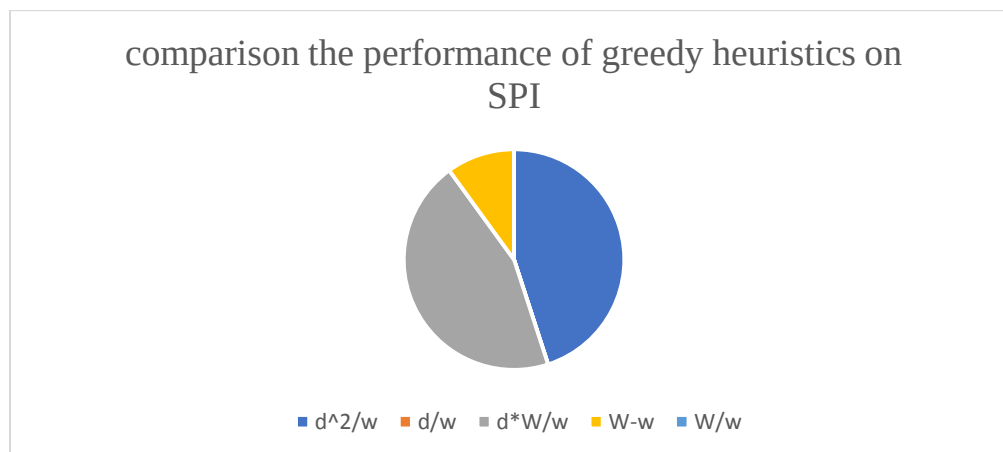
We compare the five greedy heuristics with each other and GW and GD, [15]

- (1) GW the method greedily selects the vertex with the minimum weight.
- (2) GD the method greedily selects the vertex with the maximum degree.

• Small problem instances table SPI :

| n          | m   | GW       | GD     | d/w     | d <sup>2</sup> /w | W*d/w         | W/w     | W-w           | opt    |
|------------|-----|----------|--------|---------|-------------------|---------------|---------|---------------|--------|
| 10         | 10  | 509,5    | 334.9  | 299.8   | 301.5             | <b>293.2</b>  | 308.4   | 303,3         | 284.0  |
|            | 20  | 531,6    | 437    | 412.3   | 407.7             | <b>406</b>    | 408.1   | 406           | 398.7  |
|            | 30  | 547,3    | 449    | 451.8   | <b>440.4</b>      | 444.2         | 456.6   | 442.6         | 431.3  |
|            | 40  | 564,1    | 540    | 521.2   | 510.2             | <b>508.5</b>  | 527.6   | 515.5         | 508.5  |
| 15         | 20  | 784.9    | 528.4  | 473.3   | <b>448.1</b>      | 459.1         | 482.9   | 454.2         | 441.9  |
|            | 40  | 803.5    | 648.1  | 611.2   | <b>588</b>        | 600.6         | 602.7   | 615.2         | 570.4  |
|            | 60  | 895.5    | 818.5  | 772     | 758.7             | <b>754</b>    | 774.9   | 756.9         | 726.2  |
|            | 80  | 973.1    | 880.4  | 847.8   | 830.6             | <b>830.2</b>  | 844     | 832.1         | 807.5  |
|            | 100 | 942.3    | 906.5  | 905.5   | <b>882.5</b>      | 888.7         | 923.2   | 887.9         | 880.0  |
| 20         | 20  | 1057.8   | 589.8  | 529.9   | <b>489.7</b>      | 493.4         | 533.3   | 504.1         | 473.0  |
|            | 40  | 1019.4   | 730.6  | 707.1   | 687.1             | <b>681.4</b>  | 719.2   | 690.2         | 659.3  |
|            | 60  | 1121.6   | 976.6  | 906.8   | <b>887.6</b>      | 901.2         | 920     | 909.6         | 861.8  |
|            | 80  | 1179.7   | 987.7  | 959.8   | <b>934.3</b>      | 949.3         | 962.4   | 951.2         | 898.0  |
|            | 100 | 1294.8   | 1129.7 | 1069.6  | <b>1048.1</b>     | 1061.2        | 1085.2  | 1068.5        | 1026.2 |
|            | 120 | 1258.4   | 1091.5 | 1072.7  | <b>1054.2</b>     | 1066.7        | 1098.9  | 1055.4        | 1038.2 |
| 25         | 40  | 1274.0   | 844.2  | 792.7   | 787.5             | <b>779.4</b>  | 826.7   | 797           | 756.6  |
|            | 80  | 1444.0   | 1091.9 | 1073.5  | 1055.8            | <b>1040.6</b> | 1090.2  | 1047.2        | 1008.1 |
|            | 100 | 1528.1   | 1236.9 | 1160.4  | 1147.3            | 1157.6        | 1198.1  | <b>1137.7</b> | 1106.9 |
|            | 150 | 1602.5   | 1347.7 | 1333.4  | 1292.6            | <b>1290.9</b> | 1332.3  | 1305.6        | 1264.0 |
|            | 200 | 1606.0   | 1480.1 | 1409.9  | 1401.9            | 1395.9        | 1419.4  | <b>1392.9</b> | 1373.4 |
| <b>AVG</b> |     | 1046.925 | 852.5  | 815.535 | 797.7             | 799.8         | 825.705 | 804.995       | 775.7  |

**Table2** : experimentation on SPI instance

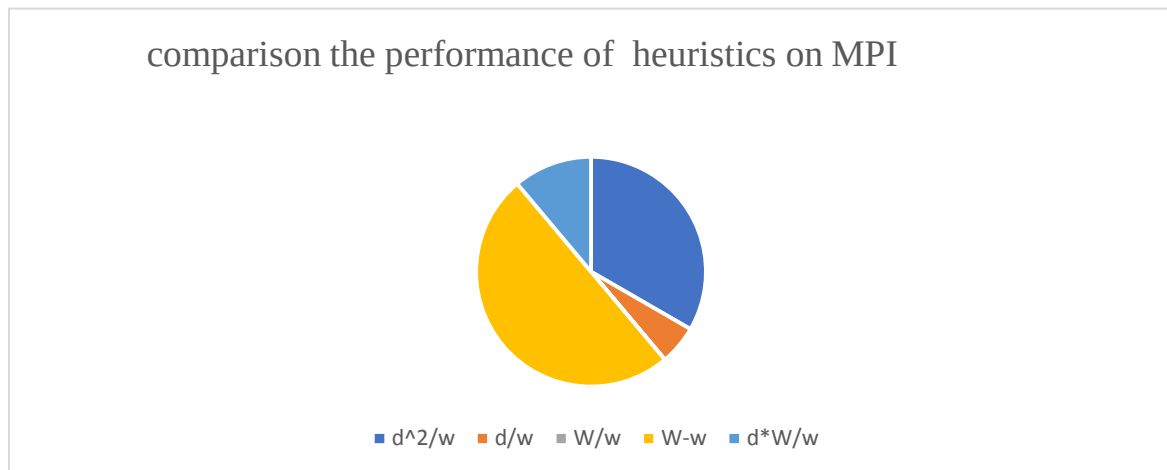


• Medium problem instances table MPI

| n   | m    | GW      | GD      | d/w           | d <sup>2</sup> /w | W*d/w         | W/w     | W-w            |
|-----|------|---------|---------|---------------|-------------------|---------------|---------|----------------|
| 50  | 50   | 2902,6  | 1467,4  | 1341,7        | 1322,8            | 1319,4        | 1359    | <b>1317</b>    |
|     | 100  | 2977,9  | 1936,8  | 1859,6        | <b>1809</b>       | 1830,5        | 1884,8  | 1823,9         |
|     | 250  | 3155,1  | 2503,8  | 2416          | 2393              | 2382,7        | 2459,2  | <b>2357,3</b>  |
|     | 500  | 3161,5  | 2902,4  | 2789,6        | 2753,7            | 2762,1        | 2783    | <b>2712,3</b>  |
|     | 750  | 3316,4  | 3083,1  | 3037,3        | <b>3009,5</b>     | 3024,2        | 30055,5 | 3030,5         |
|     | 1000 | 3413,2  | 3280,5  | <b>3229,7</b> | 3240,4            | 3256,1        | 8238    | 3240,6         |
| 100 | 100  | 6337,5  | 2926    | 2672,4        | 2612              | 2607,5        | 2733,6  | <b>2593,5</b>  |
|     | 250  | 6261,7  | 4092,1  | 3824,5        | 3745,5            | <b>3729,9</b> | 3893,5  | 3761,9         |
|     | 500  | 6567,2  | 5049,7  | 4899,4        | 4820,4            | 4827,9        | 4918,2  | <b>4794,3</b>  |
|     | 750  | 6484,9  | 5440,3  | 5299,8        | 5245,6            | <b>5215,7</b> | 5310,6  | 5225,9         |
|     | 1000 | 6830    | 5874,9  | 5749,9        | 5724,5            | 5738,1        | 5786,1  | <b>5710,4</b>  |
|     | 2000 | 6823,6  | 6367,6  | <b>6178,2</b> | 6203,6            | 6211,5        | 6281,6  | 6202,2         |
| 150 | 150  | 9213,2  | 4277,7  | 3860,2        | 3791,2            | 3775,9        | 3981,8  | <b>3763,4</b>  |
|     | 250  | 9233,4  | 5256,2  | 5011,9        | <b>4864,2</b>     | 4898,1        | 5155    | 4922,8         |
|     | 500  | 9620,6  | 6936,7  | 8965,5        | <b>8787,1</b>     | 8805,6        | 9133,4  | 8865,5         |
|     | 750  | 9920,1  | 7605    | 7294,6        | 7220,4            | 7223,4        | 7388,7  | <b>7174,8</b>  |
|     | 1000 | 9919,6  | 8052,5  | 7748,5        | 7647              | 7680,1        | 7822,1  | <b>7614,9</b>  |
|     | 2000 | 10285,6 | 9106,4  | 8924,6        | 8839,5            | <b>8836,7</b> | 8956,7  | 8838,6         |
|     | 3000 | 10123,2 | 9322,5  | 9162,8        | 9173,1            | <b>9092,6</b> | 9169,1  | 9140,7         |
| 200 | 250  | 12446,5 | 6382,5  | 5883          | <b>5735,9</b>     | 5768          | 5795,6  | 5769,7         |
|     | 500  | 12529,1 | 8137,4  | 7618,7        | 7498,4            | 7481,1        | 7712    | <b>7456,9</b>  |
|     | 750  | 13097,3 | 9328,8  | 8748,5        | 8590,8            | 8584,2        | 8788,6  | <b>8542,4</b>  |
|     | 1000 | 13041,8 | 10075,6 | 9736,9        | 9579,2            | 9572,2        | 9772,9  | <b>9525,2</b>  |
|     | 2000 | 13737,4 | 11563,1 | 11328,2       | <b>11163,7</b>    | 11180,3       | 11374,6 | 11180          |
|     | 3000 | 13684,7 | 12280,5 | 12042,3       | 12005,1           | 11989,5       | 12010,3 | <b>11921,6</b> |

|            |      |         |         |         |                |         |         |                |
|------------|------|---------|---------|---------|----------------|---------|---------|----------------|
| 250        | 250  | 15714,1 | 7194,8  | 6514,6  | <b>6337</b>    | 6352,4  | 6708,3  | 6409,3         |
|            | 500  | 16222,3 | 9536,7  | 8965,5  | <b>8787</b>    | 8805,6  | 9133,4  | 8865,5         |
|            | 750  | 16594,4 | 10937,3 | 10350,1 | <b>10139</b>   | 10218,1 | 10492,9 | 10158,2        |
|            | 1000 | 16609,2 | 11885,3 | 11288,2 | <b>11192</b>   | 11272,1 | 11470,6 | 11249          |
|            | 2000 | 16843,6 | 13936,5 | 13340,2 | 13239,5        | 13245   | 13467   | <b>13153,1</b> |
|            | 3000 | 17074,2 | 14686,2 | 14323,6 | 14248,1        | 14222,6 | 14365,5 | <b>14119,1</b> |
|            | 5000 | 16878,1 | 15558,1 | 15132,2 | 15076,3        | 15076,5 | 15203,5 | <b>15019,2</b> |
| 300        | 300  | 18867,6 | 8414,3  | 7705,9  | <b>7524,5</b>  | 7539,6  | 7801,3  | 7576,6         |
|            | 500  | 19293,1 | 10705,6 | 10014,5 | 9972,6         | 9842    | 10164,2 | <b>9836,9</b>  |
|            | 750  | 19535,9 | 12534   | 11696,1 | 11566,7        | 11562,2 | 11851,5 | <b>11539,9</b> |
|            | 1000 | 19845   | 13543,8 | 12819,8 | 12730,8        | 12683,2 | 12987   | <b>12616,4</b> |
|            | 2000 | 20220,5 | 15899,7 | 15481,6 | <b>15309,6</b> | 15324,7 | 15572,1 | 15326,4        |
|            | 3000 | 20374   | 17250,7 | 16484,1 | 16469,9        | 16544,4 | 16652,8 | <b>16465,8</b> |
|            | 5000 | 20592,6 | 18350,4 | 17884,1 | 17878,4        | 17907,8 | 18051,2 | <b>17853</b>   |
| <b>AVG</b> |      | 11788,4 | 8555,9  | 8263,30 | 8160,3         | 8163,83 | 9145,7  | <b>8013,6</b>  |

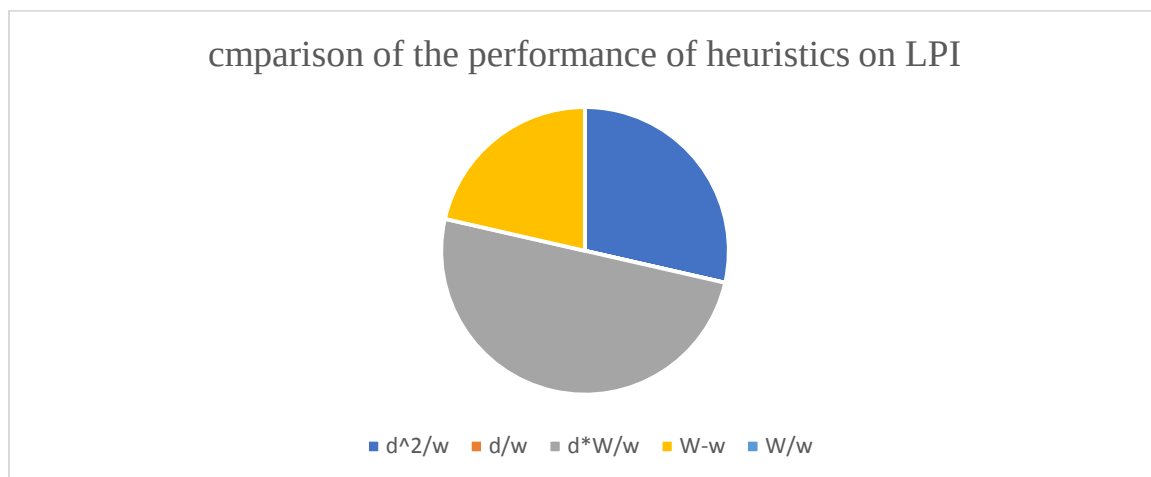
Table3 : experimentation on MPI instance



- Large problem instances table LPI

| n          | m     | d/w      | d <sup>2</sup> /w | d*W/w        | W/w      | W-w          |
|------------|-------|----------|-------------------|--------------|----------|--------------|
| 500        | 500   | 13306    | 13020             | 13038        | 13397    | <b>12949</b> |
|            | 1000  | 17109    | 17128             | 17049        | 17702    | <b>16997</b> |
|            | 2000  | 22083    | <b>21936</b>      | 21961        | 22518    | 21987        |
|            | 5000  | 28691    | <b>28330</b>      | 28514        | 29035    | 28611        |
|            | 10000 | 30768    | 30564             | 30344        | 30782    | <b>30236</b> |
| 800        | 500   | 15631    | 15341             | <b>15311</b> | 16143    | 15506        |
|            | 1000  | 23920    | 23651             | <b>23525</b> | 24719    | 23748        |
|            | 2000  | 33182    | 33061             | <b>32859</b> | 34009    | 33701        |
|            | 5000  | 40840    | 40104             | <b>39992</b> | 41035    | 40190        |
|            | 10000 | 46101    | <b>45581</b>      | 46309        | 46503    | 45679        |
| 1000       | 1000  | 25901    | <b>25235</b>      | 25605        | 26567    | 25626        |
|            | 5000  | 47586    | 47481             | <b>47410</b> | 48892    | 47867        |
|            | 10000 | 53979    | 53645             | <b>53223</b> | 54478    | 53620        |
|            | 15000 | 60177    | 61222             | <b>59399</b> | 60650    | 59827        |
|            | 20000 | 61872    | 62203             | 61818        | 62015    | 61945        |
| <b>AVG</b> |       | 36274,28 | 34566,8           | 34423,8      | 35229.66 | 34570,93     |

**Table4** : experimentation on LPI instance



### 3. Computational results of our CG algorithm:

After we seen in the previous tables, we chosen tow greedy heuristics  $G1 \arg \max_{u \in S} \frac{d(u)2}{w(u)}$  and  $G2 \arg \max_{v \in S} \frac{W(v) \times d(v)}{w(v)}$  . and we apply them in our carousel greedy algorithm CG.

There are some important parameters needed to be determined in our algorithm. Based on the comprehensive experimental results, we obtain the appropriate parameter settings.

| Parameter | Value |
|-----------|-------|
| $\alpha$  | 20    |
| $\beta$   | 5%    |

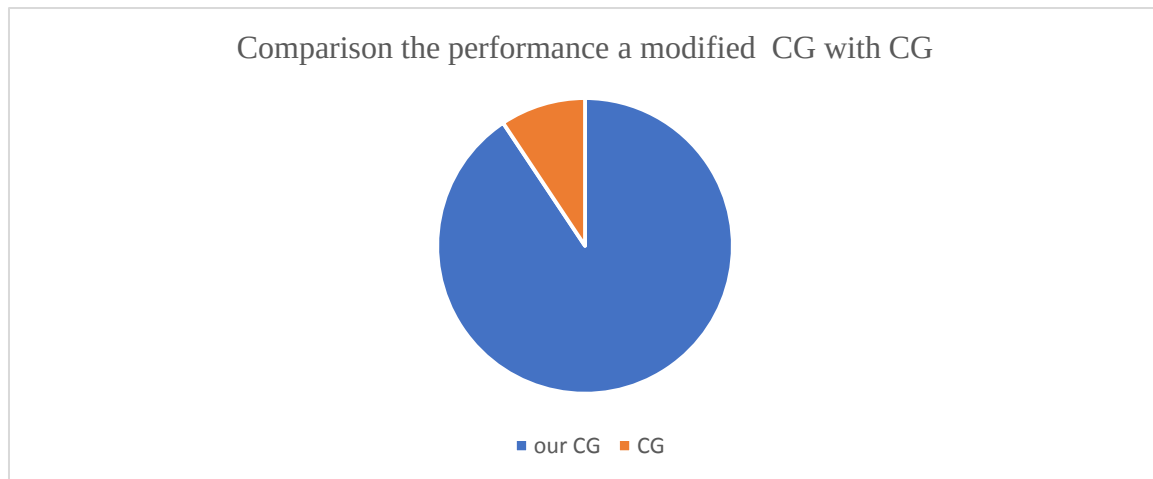
We compare our algorithm(CG modified ) with CG and several metaheuristic simulated annealing ( SA) and tabu search (TS) .[16]

- **Medium problem instances table MPI**

| n   | m    | GRDY   | CG            | Our CG        | SA     | TS     |
|-----|------|--------|---------------|---------------|--------|--------|
| 50  | 50   | 1343,7 | 1320,5        | <b>1309,7</b> | 1339,4 | 1339,4 |
|     | 100  | 1858,3 | 1801,4        | <b>1798,8</b> | 1829,6 | 1826,8 |
|     | 250  | 2416   | <b>2329,3</b> | 2355,6        | 2398,2 | 2398,2 |
|     | 500  | 2800,8 | 2742,9        | <b>2710,3</b> | 2741,7 | 2760,8 |
|     | 750  | 3052,8 | 3013,2        | <b>2983,8</b> | 3032   | 3032   |
|     | 1000 | 3234,7 | <b>3229,3</b> | 3234,4        | 3234,7 | 3234,7 |
| 100 | 100  | 2676,7 | 2653,8        | <b>2601,2</b> | 2663,2 | 2668,3 |
|     | 250  | 3816,7 | 3727,4        | <b>3701,6</b> | 3765,4 | 3765,4 |
|     | 500  | 4906,8 | 4773,6        | <b>4756,5</b> | 4791,8 | 4793,3 |
|     | 750  | 5282,7 | 5192,8        | <b>5154,7</b> | 5227,9 | 5221,7 |
|     | 1000 | 5749,9 | 5594,6        | <b>5585,7</b> | 5665   | 5649,1 |
|     | 2000 | 6184,5 | 6141,5        | <b>6139,8</b> | 6159,5 | 6170,5 |
| 150 | 150  | 3864,1 | 3808,4        | <b>3748,9</b> | 3834,2 | 3837,4 |
|     | 250  | 5019   | 4926,7        | <b>4870,7</b> | 4953,3 | 4951,8 |
|     | 500  | 6551,6 | <b>6357,5</b> | 6359,4        | 6449   | 6452,5 |
|     | 750  | 7291,3 | 7131,1        | <b>7111,2</b> | 7220,3 | 7222,4 |
|     | 1000 | 7747,7 | <b>7536,3</b> | 7558          | 7623,9 | 7614,8 |

|     |      |         |                |                |         |         |
|-----|------|---------|----------------|----------------|---------|---------|
|     | 2000 | 8900,8  | 8744,5         | <b>8720</b>    | 8767,3  | 8767,6  |
|     | 3000 | 9178,9  | 9033,9         | <b>9008,7</b>  | 9070,2  | 9071,7  |
| 200 | 250  | 5892,9  | 5808,7         | <b>5691</b>    | 5841,7  | 5841,7  |
|     | 500  | 7620,8  | 7444,8         | <b>7386,9</b>  | 7521,4  | 7518,6  |
|     | 750  | 8721,2  | <b>8466,7</b>  | 8480,1         | 8590,9  | 8604,9  |
|     | 1000 | 9729,2  | 9432,5         | <b>9356,5</b>  | 9537,6  | 9507,7  |
|     | 2000 | 11341   | 11042,1        | <b>10990</b>   | 11076,8 | 11076,6 |
|     | 3000 | 12027,5 | <b>11803,8</b> | 11814          | 11829,5 | 11834,7 |
| 250 | 250  | 6508,3  | 6421,5         | <b>6324,3</b>  | 6475,8  | 6475    |
|     | 500  | 8960,3  | 8796,5         | <b>8712,5</b>  | 8874,9  | 8869,4  |
|     | 750  | 10342,4 | 10125,5        | <b>10054,9</b> | 10205,6 | 10211,8 |
|     | 1000 | 11313   | 11083,9        | 11104,3        | 11183,4 | 11165,5 |
|     | 2000 | 13361,7 | 13081,9        | <b>13076,1</b> | 13110,3 | 13097,4 |
|     | 3000 | 14320   | 14032,7        | <b>13990,4</b> | 14058   | 14056,9 |
|     | 5000 | 15045,9 | <b>14830,5</b> | 14835,8        | 14957,8 | 14940,1 |
| 300 | 300  | 7695,6  | 7579,2         | <b>7507,3</b>  | 7634,9  | 7634    |
|     | 500  | 10011,7 | 9750,5         | <b>9733,5</b>  | 9844,5  | 9844,4  |
|     | 750  | 11700,6 | 11407,9        | <b>11400,4</b> | 11570,5 | 11565,6 |
|     | 1000 | 12825,5 | 12517,5        | <b>12468,5</b> | 12618   | 12618,7 |
|     | 2000 | 15467,4 | 15091,3        | <b>15055,7</b> | 15211,5 | 15201,5 |
|     | 3000 | 16495,4 | 16208,7        | <b>16177,1</b> | 16265,5 | 16243,2 |
|     | 5000 | 17935   | <b>17563,4</b> | 17631,5        | 17681,6 | 17648   |

Table5: experimentation on MPI instance



We can notice clearly that there are great difference when we compare our algorithm with CG where the solution quality are extremely better .

*General conclusion*

## **General conclusion**

The minimum weight vertex cover problem is one of the hardest NP-complete problems that has its real applications everywhere.

The question is how to find a good solution within respected times, and for this reason we have done this work.

The objective of this thesis was a proposition of an approach to tackle this problem giving comparable results with the existent state of art. using the strong C++language we implemented this approach where we confronted many problems concerning how to really learn and apply the basics of the carousel strategy and the greedy algorithm especially in how to get innovated for the creation of construction procedures. and how to obtain minimum running time. we were able to show that CG overcomes some of the major limitations of traditional greedy algorithms.

Though the good results of this approach, it stills missing a convenient behavior because it is possible for CG to remove and later select the same element. If this happens for a full turn of the carousel at iteration  $i$ , then the remaining  $\alpha-i$  iterations are unnecessary. In this case, we say that CG produces a loop or a cycle. in future work, it should to investigate ways to quickly detect and avoid loops.

We aim in the near future to hybridize the CG with a metaheuristic to obtain new metaheuristic.

## Reference :

- [1] : S. Bouamama, C. Blum, and A. Boukerram. A population-based iterated greedy algorithm for the minimum weight vertex cover problem. *Applied Soft Computing*, 12(6):1632–1639, 2012.
- [2] : A.Potluri ,A.Singh .Hybrid metaheuristic algorithm for minimum weight dominating set *Applied soft computing* 13(2013) 76-88
- [3]: S.G Shrinivas et .al . application of graph theory in computer science an overview *International journal of engineering science and technology*. 2010 ,4610-4621.
- [4] : Robin j. Wilson *Introduction to graph theory* fourth edition
- [5] : Keijo Ruohonen graph theory lecture .[http://math.tut.fi/~ruohonen/GT\\_English.pdf](http://math.tut.fi/~ruohonen/GT_English.pdf).
- [6] : lecture intro to algorithm MIT open course ware <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science>.
- [7] lecture basic problem of Graph Theory web site :  
[http://www.math.uni.lodz.pl/~marmaj/MBL/Lectures/9\\_basic\\_problems.pdf](http://www.math.uni.lodz.pl/~marmaj/MBL/Lectures/9_basic_problems.pdf)
- [8] : Eugene. L .Lawler *book combinatorial optimization : network and matroids*
- [9] : lecture heuristic optimization methods : introduction to optimization university of zagreb  
[https://www.fer.unizg.hr/download/repository/Lecture\\_1-2\\_%5BENG%5D\\_-\\_Introduction\\_to\\_optimization.pdf](https://www.fer.unizg.hr/download/repository/Lecture_1-2_%5BENG%5D_-_Introduction_to_optimization.pdf) .
- [10] : lecture note : discrete optimization Utrecht University Academic Year 2011/2012.  
[http://wwwhome.math.utwente.nl/~kernw/lectures/DO/DO\\_script.pdf](http://wwwhome.math.utwente.nl/~kernw/lectures/DO/DO_script.pdf).
- [11] : F. Rothlauf, *Design of Modern Heuristics*, Natural Computing Series, DOI 10.1007/978-3-540-72962-4 2, © Springer-Verlag Berlin Heidelberg 2011
- [12]: lecture note combinatorial optimization: exact and approximate algorithm.  
<http://theory.stanford.edu/~trevisan/books/cs261.pdf>.
- [13] : lecture approximate algorithm spring 2015 MIT open course ware  
<https://ocw.mit.edu/courses/electrical-engineering-and-computer-science>.
- [14] : V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.
- [15] : S.SHYU An ant colony optimization algorithm for the minimum weight vertex cover problem . *Annals of operations research* 131,283-304,2004.

## *Reference*

---

- [16]: Carmine Cerrone, Raffaele Cerulli, Bruce Golden Carousel Greedy: A Generalized Greedy Algorithm with Applications in Optimization. [10.1016/j.cor.2017.03.016](https://doi.org/10.1016/j.cor.2017.03.016).
- [17]: Application of vertex cover <http://www.dharwadker.org/pirzada/applications>.
- [18] : book Introduction to algorithm second edition Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein .

## **Abstract**

In this study we deal with minimum weight vertex cover problem as one of the fundamental problems in graph theory with many real-life applications such as ,in wireless communication ,circuit design and network flows .It is well-known NP-complete problem and hence no polynomial-time algorithm has been found yet for solving it to optimality.

We have implemented a modified carousel algorithm for tackling this problem in order to obtain good feasible solutions in reasonable computational time . The letter is enhanced by introducing greedy heuristics to improve the quality of solution. The performance of our approach has been tested on well-known dataset.

**Keywords :**NP-complete problem, greedy algorithm ,minimum weight vertex cover problem, heuristic.

## **Résumé:**

Dans cette étude, nous traitons le de la couverture par les sommets de poids minimum qui est un problème fondamental dans la théorie de graph ,couvrant de nombreuses applications dans les domaines de la communication sans fils, de la conception des circuits et du contrôle du flux dans les réseaux .Ce problème appartient à la classe des problèmes NP-complete ou aucun algorithme polynomial n'a encore été trouve pour le résoudre à l'optimalité .

Nous avons adapté Carousel -algorithme pour traiter ce problème afin d'obtenir des bonnes solutions dans un temps de calcul raisonnable. Ce dernier est renforcé par l'introduction des heuristiques gloutons pour améliorer la qualité des solutions générées .La performance de notre approche a été testée sur des jeux de données bien connues.

**Mots-clés :** NP-complet, algorithme glouton, Problème du transversal de poids minimum  
Heuristique.