PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
MOHAMED BOUDIAF UNIVERSITY OF M'SILA

FACULTY OF MATHEMATICS AND INFORMATICS
DEPARTMENT OF COMPUTER SCIENCE

Thesis submitted for the degree of Master in Computer Science

DOMAIN: Mathematics and Informatics

FILIERE: Informatics

OPTION: Informatique Décionnelle et Optimisation (IDO)

By: BADA HOUSSAM

Title

Community Detection Algorithm Based On Discrete Particle Swarm Optimization

Defended publicly on: 12 / 06 /2024 to the jury composed of :

Dr. Salah Guesmia	University of M'sila	President
Pr. Allaoua Hemmak	University of M'sila	Supervisor
Dr. Azeddine Attir	University of M'sila	Examiner

Academic year: 2023 / 2024

DEDICATIONS

"I dedicate this modest work to :

My parents

May they find here the testimony of my deep gratitude and acknowledgment

I wish them good health and long life

To my brothers and my sisters, my grandparents and my family who give love and liveliness.

To all those who have helped me - directly or indirectly - and those who shared with me the emotional moments during the accomplishment of this work and who warmly supported and encouraged throughout my journey.

To all my friends who have always encouraged me, and to whom I wish more success.

Thanks!"

Houssam Bada

ACKNOWLEDGMENTS

Praise be to the Almighty God who has given me faith, courage, and patience to carry out this modest work.

I want to express my deep gratitude to my supervisor **Pr. Allaoua Hemmak** for the confidence he has placed in me, through his presence always with me, by his direction, his modesty, his advice, and constructive remarks for the good progress of this work.

I like to express my appreciation to the members of the jury **Dr. Salah Guesmia** and **Dr. Azeddine Attir**, for their time and effort in carefully reviewing and evaluating our thesis.Their valuable feedback and experience will help us improve and refining our research.

I thank **TAIBI Salah Eddine** who encouraged me to do this research.I wish him all the best and good health.

I would like to thank everyone who helps me to improve my modest work. and who gave me any remark that helped me to perfect this manuscript.

I express my deep gratitude to my parents, my brothers, my sisters, and my whole family for their encouragement and prayers that allowed me to achieve this modest job. I am very grateful for the confidence they have placed in me.

Finally, I express my gratitude to all those who have contributed in one way or another to the development of this work.

O Allah, send your blessings on your noble messenger, his family, and companions, and bless us in our life.

Contents

List of Figures

List of Tables

List of Algorithms

General introduction

Context

Networks are widely used in fields such as computer science, physics, and mathematics to model various types of complex systems. Examples of these systems include biological networks, technological networks, social networks, and political election networks. Mathematically, a network can be represented as a graph where vertices signify the network objects, and edges represent the relationships between them.

One interesting property of complex networks that has attracted considerable attention from researchers across different disciplines is the community structure. A community is a subset of nodes within a graph where the connections between these nodes are denser than the connections with the rest of the network. Community detection methods have numerous practical applications, including cancer detection, product recommendation, link prediction, and software package refactoring.

In recent years, numerous approaches have been proposed for community detection [27–30]. Generally, community detection can be framed as an optimization problem, where the objective is to find an optimal solution according to a predefined objective function. Often, optimizing this objective is NP-hard. Consequently, many studies have focused on using metaheuristic methods such as genetic algorithms, simulated annealing, and collaborative evolutionary algorithms to tackle this challenge.

In addition to EA-based metaheuristic optimization techniques, another significant class is swarm intelligence-based methods, with particle swarm optimization (PSO) being a prominent example. PSO is inspired by the social behavior of animals, such as fish schooling and bird flocking. It optimizes a problem by utilizing a group of particles, each representing a candidate solution. These candidate solutions are updated using simple rules learned by the particles. Due to its effectiveness and ease of implementation, PSO has become highly popular in the optimization field, leading to the development of numerous variants. However, canonical PSO is specifically designed for continuous optimization problems.

Motivation

The motivation behind this scientific report is driven by our keen interest and passion for community detection within complex networks, specifically using the Particle Swarm Optimization (PSO) algorithm. With the growing demand for effective community detection systems across various domains, we recognize the importance of leveraging PSO to address the challenges of identifying communities and developing accurate, efficient solutions. Our primary objective is to explore and implement effective techniques in community detection based on PSO, evaluating their performance and effectiveness.

Through this project, we aim to provide valuable insights, recommendations, and contributions to the field of community detection. By conducting thorough research and experimentation, we seek to develop robust and reliable community detection systems that can cater to the diverse needs of different domains, ultimately pushing the boundaries of what is possible in this field and paving the way for future innovations.

Objectives

The main objective of this thesis is to design an easy and efficient approach to community detection in complex networks, based on the Discrete Particle Swarm Optimization algorithm without prior knowledge about the size of communities and the number of communities. The proposed approach is evaluated on different types of networks and its performance is compared with the other community detection algorithms.

Thesis organization

This thesis is organized into five primary chapters, each focusing on different aspects of graph theory, combinatorial optimization, and community detection within complex networks.

The first chapter, "**Fundamental Notions of Graph Theory**" provides a comprehensive introduction to graph theory, covering its applications, essential definitions, and various types of graphs such as complete graphs, subgraphs, and bipartite graphs. It also delves into key terminologies like adjacency and incidence, degrees, and measures of graph similarity and centrality. This section concludes with graph representation techniques and addresses some common problems in graph theory, setting a solid foundation for understanding the subsequent sections.

The second chapter, "**Overview of Combinatorial Optimization**" shifts focus to the optimization landscape. It begins by explaining the importance of combinatorial optimization and the processes involved in solving optimization problems. This section defines optimization

problems formally, discusses the concepts of global optima and search spaces, and classifies different types of optimization problems. Additionally, it examines algorithmic efficiency, the complexity classes, and various optimization methods, including deterministic and probabilistic algorithms, and approaches for solving combinatorial optimization problems. This section equips readers with the necessary tools and methodologies for tackling complex optimization tasks.

The third chapter, "**Community Detection**" concentrates on the main focus of the thesis: detecting communities within networks. It starts with an introduction to the concept of communities and their structures, followed by a detailed discussion on the objectives and applications of community detection. Various methods and algorithms for community detection are classified and evaluated, along with measures to assess the quality of detected communities.

The fourth chapter, "**Community Detection Algorithm Based on Discrete Particle Swarm Optimization**" presents a specific algorithm developed . It includes an introduction, objectives, related work, and fundamental principles of the PSO algorithm. The proposed algorithm is detailed, covering simple discrete particle swarm optimization, discrete particle swarm optimization with a redefined operator, and a modified IDPSO-RO, which includes modularity density, local search strategy, majority voting, and the algorithm of the modified IDPSO-RO.

The fifth and final chapter, " **Implementation and Experimental Results**" describes the practical aspects of the research. It begins with an introduction to the working environment, detailing the hardware and software environments, platform and IDE, and libraries used. The experimental results and analysis follow, showcasing the performance of the proposed algorithm on various real-world networks, including the Karate Club network, Dolphin network, American Politics Books network, and the American Football network. This section concludes with a comparison of the results, highlighting the effectiveness and efficiency of the proposed community detection algorithm.We close our work with a general conclusion and a look to the future.

By exploring the fundamental concepts, specific techniques, and practical implementation of community detection based on discrete particle swarm optimization, our dissertation aims to contribute to the advancement of this field

Chapter 1

Fundamental notions of graph theory

Graphs serve as fundamental models for various natural and human-mad structures, finding application across computer science, physical sciences, biology, and social systems. They offer a versatile framework for representing relationships and dynamic processes. Indeed, graph theory finds relevance in addressing numerous practical problems due to its broad applicability across different disciplines [31].

1.1 Application of graph theory

Graph theory is applied across a wide range of fields, including computer science, biology, sociology, and transportation, due to its versatility in representing and analyzing complex relationships and systems through graph-based structures [32]:

1. **Computer Science:**
   Graph theory is essential for designing computer networks, developing routing algorithms, and modeling relationships in databases. Additionally, it underpins various computer science algorithms like graph traversal methods.

2. **Biology:** Graph theory aids in modeling biological networks and analyzing complex processes like gene regulation and metabolic pathways. It also supports the analysis of evolutionary relationships represented as phylogenetic trees in phylogenetics.

3. **Sociology:**
   Graph theory plays a key role in social network analysis, exploring the structure and dynamics of social networks to understand interactions and information flow. Additionally, it models the spread of opinions, behaviors, and ideas within these networks.

4. **Transportation:**
   Graph theory is vital for optimizing transportation networks, aiding in route planning, traffic management, and resource allocation. Additionally, it enhances supply chain management and delivery scheduling in logistics operations.

5. **Other Fields:**

   - **Chemistry:** Representing chemical compounds and reactions as molecular graphs facilitates analysis, contributing to drug discovery and materials science.
   - **Finance:** Graph theory finds application in financial networks, portfolio optimization, and risk management, enabling the examination of interconnections and systemic risks within financial systems.

1.2 Graphs

A graph comprises a collection of vertices (also known as nodes or points) and a set of edges (or lines/arcs) connecting pairs of vertices. Typically, vertices represent objects, while edges signify relationships between these objects. Therefore, a graph serves as a model for representing information involving objects and their interconnections.

As in Figure. 1.1. The A , B, C, D and E are called vertices, the lines are called edges, and the whole diagram is called a graph.

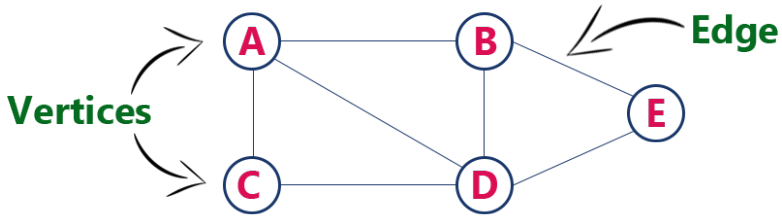


Figure 1.1: Graph [1]

1.3 Definitions

1.3.1 Simple Graph

A simple graph is defined as a graph where each pair of vertices is connected by at most one edge, and no edge connects a vertex to itself [33].For example Figure. 1.1

1.3.2 Un-directed graph

An undirected graph, denoted as G, is composed of a set V of vertices and a set E of edges. Each edge in E ($e \in E$) connects an unordered pair of vertices, meaning there is no direction associated with the edges. [34].

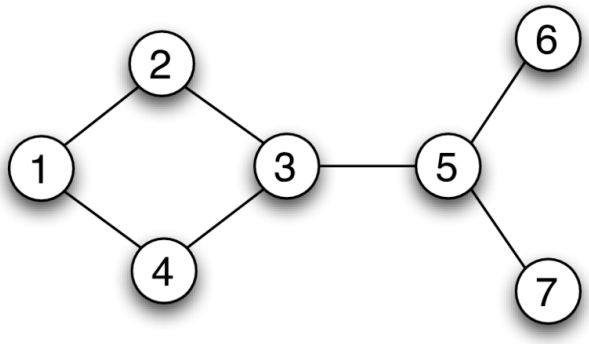


Figure 1.2: An undirected graph with 7 nodes and 7 edges [2]

1.3.3 Directed graph

A directed graph, also known as a digraph G, comprises a set V of vertices and a set E of edges. Each edge in E ($e \in E$) is linked to an ordered pair of vertices, indicating a specific direction. Thus, if the edges in the graph G have directions, it is referred to as a directed graph [34].

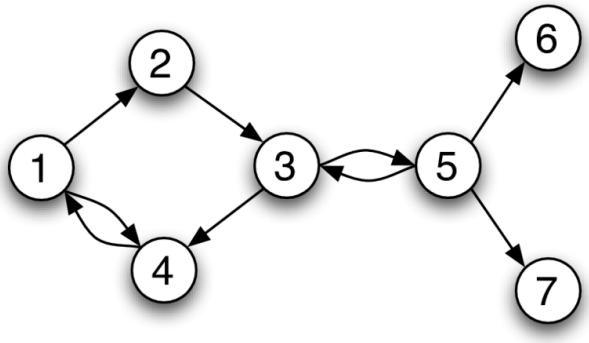


Figure 1.3: A directed graph with 7 nodes and 9 edges [3]

1.4 Types of graphs

1.4.1 Complete Graphs

A complete graph is a simple graph where every vertex is connected to every other vertex (distinct vertices) by a single edge. A complete graph with n vertices is denoted as Kn [35]. Below are a few examples of completed graphs.

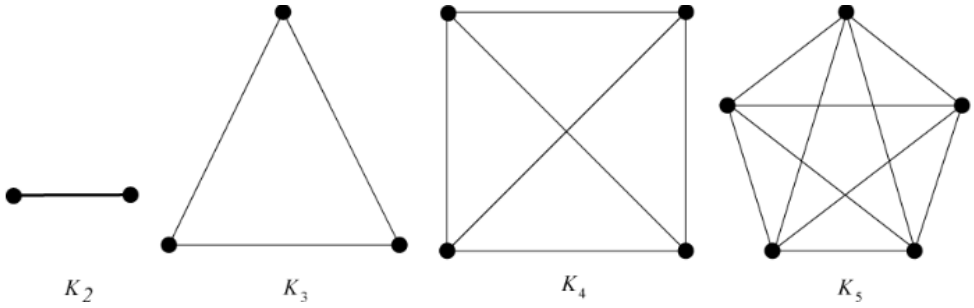


Figure 1.4: Complete Graph Example [4]

1.4.2 Subgraphs

A subgraph of a graph $G = (V, E)$ is a graph $G' = (V', E')$ where the set of vertices V' is a subset of V ($V' \subseteq V$), and the set of edges E' is a subset of E ($E' \subseteq E$) [5].

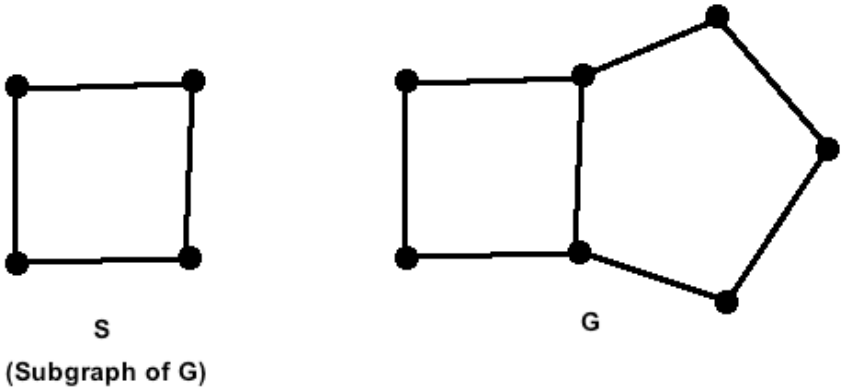


Figure 1.5: Subgraph [5]

1.4.3 Null Graphs

A graph that has no edges is referred to as a null graph. A null graph with n vertices is represented as Nn [36].

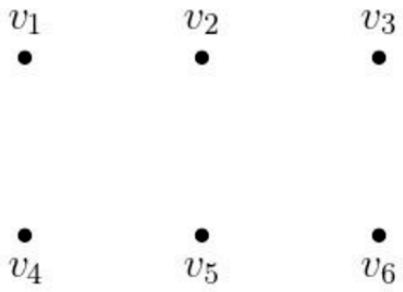


Figure 1.6: A null graph N6 with six vertices [6]

1.4.4 Multigraphs

If a graph permits multiple edges between its vertices, it is referred to as a multigraph. [7]

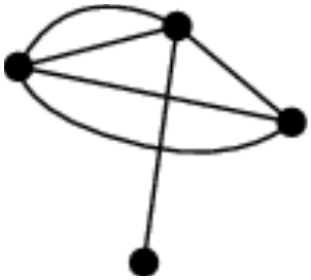


Figure 1.7: A multigraph [7]

1.4.5 Pseudo graph

A pseudograph is a graph G that includes self-loops and allows multiple edges between vertices. It is characterized by the presence of loops (edges connecting a vertex to itself) and the possibility of having more than one edge connecting two vertices [37].

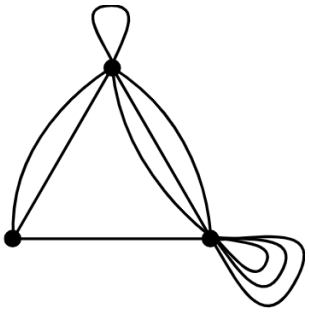


Figure 1.8: Pseudo graph [8]

1.4.6 hypergraph

If edges are defined as arbitrary subsets of vertices, rather than pairs, the resulting structure is known as a hypergraph(Figure. 1.9) [9].

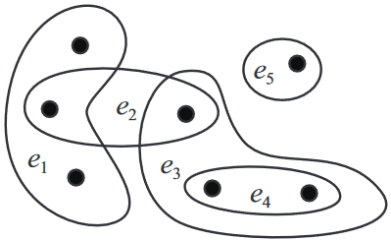


Figure 1.9: A hypergraph with 7 vertices and 5 edges. [9]

1.4.7 trivial graph

A graph that consists of a single vertex is considered trivial.

1.4.8 Bipartite Graphs

A graph G is termed a bipartite graph if its vertex set V can be divided into two separate and independent sets, V_1 and V_2. These sets are referred to as the partite sets of G. In a bipartite graph, each edge connects a vertex from set V1 to a vertex in set V2 [38].

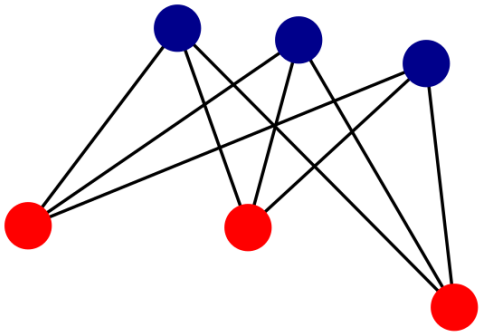


Figure 1.10: A bipartite graph [10]

1.5 Terminologies and basic notation

1.5.1 Adjacency and incidence and neighborhood:

In a graph G, if two vertices are linked by an edge, they are considered **adjacent**. In our graph example(Figure. 1.11), vertex v_1 has two adjacent vertices, v_2 and v_3 [11].

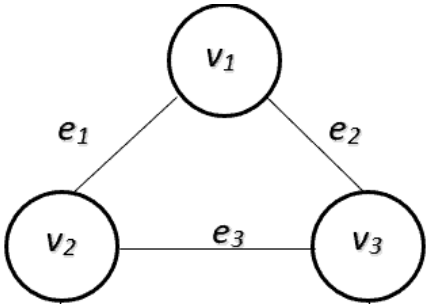


Figure 1.11: Illustrate adjacency notation [11]

In a graph G,two edges are considered **incident** if they have a common vertex. For example(Figure. 1.11), edge (v_1, v_2) and edge (v_1, v_3) are incident as they share the same vertex v_1.

The set of vertices adjacent to v is termed the **neighborhood** of v, represented as N(v). To differentiate it from the **closed neighborhood**, denoted as $N[v] = N(v) \cup v$, N(v) is sometimes referred to as the **open neighborhood** of v [39].

1.5.2 Degree

The **degree** of a vertex v in a graph G, represented as deg(v) or d(v), refers to the number of edges incident to v in G, counting each loop at v twice. A vertex with a degree of 0 is termed an **isolated** vertex, while a vertex with a degree of 1 is known as a **pendant** vertex. [40].

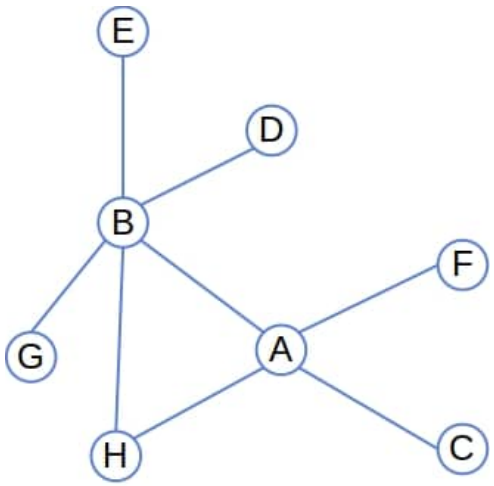


Figure 1.12: Simple graph [12]

For example(Figure. 1.12), the degree of A is 4(deg(A)=4), and the degree of B is 5 (deg(B)=5) in this graph since they have 4 and 5 neighbors, respectively [12].

The maximum degree of a graph G, represented by $\Delta(G)$, is the highest degree among all the vertices in G [41].

$$\Delta(G) = \max\{\deg(v) \mid v \in V(G)\}$$

Similarly, we define **the minimum** degree of a graph G and denote it by $\delta(G)$ [41].

$$\delta(G) = \min\{\deg(v) \mid v \in V(G)\}$$

The maximum and the minimum degree of the graph in Figure. 1.12 are $\Delta(G) = 5, \delta(G) = 1$

1.5.3 Walks, Trails, Paths, cycles

A walk in a graph G is a sequence, denoted as $W : v_0e_1v_1e_2v_2 \dots e_pv_p$, that alternates between vertices and edges. It starts and ends with vertices, where v_{i-1} and v_i are the endpoints of edge e_i. Here, v_0 is the starting point, and v_p is the endpoint of the walk. If v_0 equals v_p ($v_0 = v_p$), the walk is closed; otherwise, it's open. The length of a walk is determined by the number of edges it contains. A walk with a length of 0 consists of a single vertex [13].

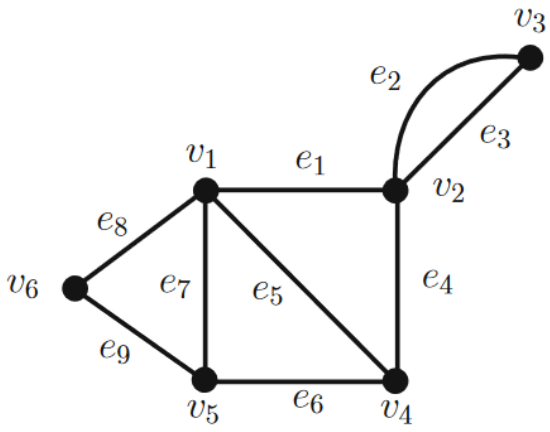


Figure 1.13: Graph illustrating walks, trails, paths, and cycles [13]

In the graph of Figure. 1.13, $v_5e_7v_1e_1v_2e_4v_4e_5v_1e_7v_5e_9v_6$ is a **walk** .
$v_1e_1v_2e_2v_3e_3v_2e_1v_1$ is a **closed walk**.

A walk is termed a **trail** if each edge it includes is unique. In other words, a walk becomes a trail when each edge is traversed at most once. Continuing from the previous Figure. 1.13 $v_1e_1v_2e_4v_4e_5v_1e_7v_5$ is a trail.

A trail is referred to as a **path** if all its vertices are unique. In other words, a trail becomes a path when each vertex is visited at most once, except possibly for the initial and terminal vertices, which may be the same.
Continuing from the previous Figure. 1.13 $v_6e_8v_1e_1v_2e_2v_3$ is a path, $v_6v_1v_2v_3$ is a path.

A **cycle** is a closed trail where all vertices are unique(closed paths).
Continuing from the previous Figure. 1.13 $v_1e_1v_2e_4v_4e_6v_5e_7v_1$ is a cycle.and $v_1v_2v_4v_5v_6v_1$ is a cycle.

1.5.4 Density of a graph

The density of a graph G = (V, E) is a measure of its connectivity, indicating the number of links within a network or graph. The denser the graph, the more connected it is [42, 43]. To measure density in a graph, we divide the number of observed edges (the total number of connections) by the maximum number of possible edges (the number of possible connections) [42].Therefore, it varies between 0 for an empty graph (the vertices are isolated) and 1 for a complete graph (there is a link between each pair of vertices). It is calculated by the formula [43]:

$$D = \frac{|E|}{|V|(|V|-1)/2}$$

where :
|E| : the number of edges in the graph.
|V | : the number of vertices in the graph.

1.5.5 Clique

A clique, as depicted in Figure. 1.14, is a collection of nodes (vertices) where every pair of nodes is connected by an edge. A clique is considered maximal if it cannot be included within a larger clique [44, 45].

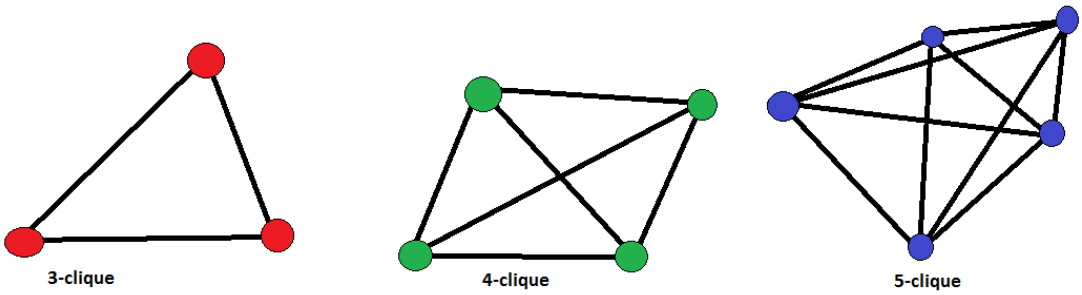


Figure 1.14: K-clique example [14]

1.5.6 Measures of graph similarity and centrality

1.5.6.1 Measures of similarity

Let two nodes i and j with N_i and N_j, represent the set of neighbors of i and the set of neighbors of j respectively.

The similarity between two nodes can be calculated using a number of methods:

1. **Jaccard index:** For two nodes i and j, the Jaccard index is defined by [46] :

$$Jaccard(i,j) = \frac{|N_i \cap N_j|}{|N_i \cup N_j|}$$

where : N_i is the set of neighbors of i.
Nj is the set of neighbors of j.
$|N_i \cap N_j|$ is the number of common neighbors of both nodes i and j.
$|N_i \cup N_j|$ is the total number of neighbors of both nodes i and j.

2. **Cosine similarity:** is measured using the following formula [47] :

$$\text{Cos}(i,j) = \frac{|N_i \cap N_j|}{\sqrt{|N_i|.|N_j|}}$$

Where:
$|N_i \cap N_j|$ is the number of common neighbors of nodes i and j.
$|N_i|.|N_j|$ is the multiplication of the neighbors of i and j.

1.5.6.2 Centrality measure

In order to analyze the positions of individuals relative to others in a graph, centrality measures are used to characterize them. Several types of centrality have been defined, including the following:

1. **Degree centrality:** The most straightforward measure of centrality is degree centrality. It's determined by the number of links connected to a node, commonly referred to as its degree or the count of adjacent neighbors. [48].

2. **Closeness Centrality** It indicates how close the vertex is to all the vertices in the graph, and how quickly it can interact with these vertices. It is calculated as the inverse of the sum of the shortest path lengths between the node and all the other nodes in the graph. Thus, the more central a node, the closer it is to all other nodes [48, 49], it is formally written as :
$$C_c(v) = \frac{1}{\sum_{u \in V \setminus \{v\}} d_G(u, v)}$$
where $d_G(u, v)$ is the distance between vertices u and v, such as the number of edges in the shortest path between two vertices or the sum of the valuations of these edges for valuated graphs.

3. **Betweenness Centrality:** Betweenness centrality is a critical concept in network analysis, assessing the significance of a vertex in information transmission within a graph. A vertex assumes a central role if numerous shortest paths between pairs of vertices traverse through it. This centrality metric quantifies the frequency with which the vertex appears on the shortest path between any pair of other nodes in the graph. [42, 50], and is written as follows:
$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$
Where:
σ_{st} is the total number of shortest paths from node s to t.
$\sigma_{st}(v)$ is the number of paths between s and t that pass through v.

   The centrality $C_B(V)$of a node is high, if most of the graph's communiacations pass through it.

4. **PageRank centrality:** The PageRank measure was introduced in the late 1990s by computer scientists Brin and Page [51] to rank web pages. It uses user behavior while brows-

ing the Web to rank pages, where pages are represented as graph nodes and hyperlinks as edges. PageRank indicates the importance of nodes based on the principle that a node's importance is the expected sum of the importance of all its connected nodes (neighbors) and the direction of the edges. Its value corresponds to the probability distribution of random access to nodes. In graph theory, PageRank recursively calculates a normalized and propagated value for each node in a graph [52]. Given x and p as two nodes in a graph G, the PageRank of x is given by:

$$\mathrm{PR}(x) = (1-c) + c * \sum_{p \in Pnt_{\mathrm{in}}(x)} \frac{\mathrm{PR}(p)}{|Pnt_{\mathrm{out}}(p)|}$$

where c is a damping factor with a typical value of 0.85, $Pnt_{\mathrm{in}}(x)$ is the set of nodes pointing to x, $Pnt_{\mathrm{out}}(p)$ is the set of nodes pointed to by p, and $|P_{\mathrm{out}}(p)|$ is the cardinality of that set. PageRank operates on a directed graph, and the value for a given node is calculated iteratively based on the PageRank of the nodes pointing to it [53].

1.6 Graph Representation

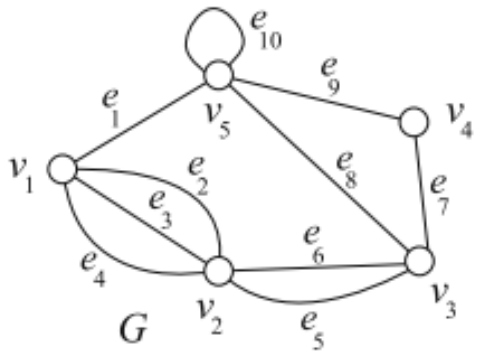
When discussing graph problems, we primarily use two graph representations: the adjacency list and the adjacency matrix. Understanding the differences between these two representations is crucial for effectively addressing graph-related challenges.

1.6.1 Adjacency Matrix

Consider a graph G with a vertex set $V = \{v_1, v_2, ..., v_n\}$ and an edge set $E = \{e_1, e_2, ..., e_m\}$. The adjacency matrix $A(G)$ of G is an $n \times n$ matrix represented as $A(G) = [a_{ij}]$, where a_{ij} denotes the number of edges between the two vertices v_i and vj. [41]

	v_1	v_2	v_3	v_4	v_5
v_1	0	3	0	0	1
v_2	3	0	2	0	0
v_3	0	2	0	1	1
v_4	0	0	1	0	1
v_5	1	0	1	1	1

(a) Adjacency matrix representation of G



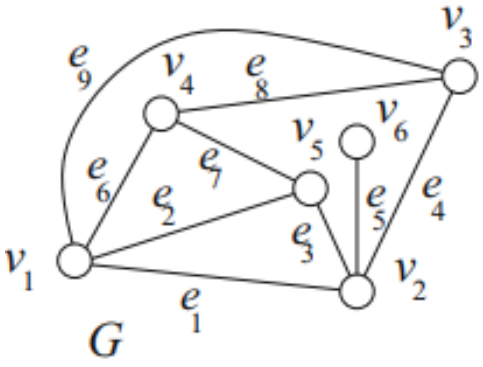
(b) A graph G [41]

Figure 1.15: Matrix and A graph G

In a simple graph G, each entry of its adjacency matrix A(G) is either zero or one, and the main diagonal of the matrix exclusively comprises zeros. [41].

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	1	1	1	1	0
v_2	1	0	1	0	1	1
v_3	1	1	0	1	0	0
v_4	1	0	1	0	1	0
v_5	1	1	0	1	0	0
v_6	0	1	0	0	0	0

(a) Adjacency matrix representation of G



(b) A simple graph G [41]

Figure 1.16: Matrix and A simple graph G

1.6.2 Adjacency list

An adjacency list is a collection of lists used to represent a graph, where each node maintains a list of its adjacent vertices. Each list corresponds to a vertex u, and includes the edges (u, v) originating from u [54].

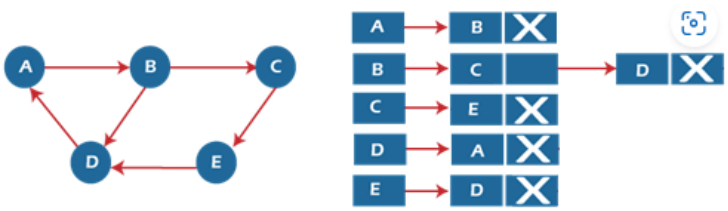


Figure 1.17: Adjacency list [15]

1.6.3 Comparison

An adjacency matrix requires $\theta(V^2)$ storage space (where the constant factor is small due to each entry being just a bit). On the other hand, an adjacency list occupies $\theta(V + E)$ space.

The storage requirement for an adjacency list is worst-case when the graph is dense, meaning $E = \theta(V^2)$. In this scenario, the space complexity matches that of the adjacency matrix representation. However, the $\theta(V + E)$ space complexity of the adjacency list is generally more efficient for the general case. Additionally, adjacency lists provide faster access to the set of adjacent vertices for a given vertex compared to an adjacency matrix, with a time complexity of $O(neighbors)$ for the former and $O(V)$ for the latter [55].

1.7 Some problems of graph theory

1.7.1 Graph Partition

A graph partition involves reducing a graph to a smaller graph by dividing its set of nodes into distinct groups. The edges of the original graph that connect nodes from different groups remain in the partitioned graph. If the number of resulting edges is significantly smaller than that of the original graph, the partitioned graph may offer advantages for analysis and problem-solving. Finding an optimal partition that simplifies graph analysis is a challenging task, but one with numerous applications in scientific computing, and task scheduling on multiprocessor computers, among others [56]. In recent years, the graph partition problem has gained significance due to its utility in clustering and identifying cliques in social, pathological, and biological networks. Two common examples of graph partitioning are the minimum cut and maximum cut problems.

So a graph partition problem is to cut a graph into 2 or more ”good” pieces.Note that not all graphs have good partitions.
Question: Can we certify that there are no good clusters in a graph?
”Good” clusters have the following properties [57]:

- internally (intra) - well connected.
- externally (inter) - relatively poor.

Graph partitioning is a challenging problem(NP-hard problem), often addressed using heuristic methods due to its complexity. These methods generally fall into two categories: local and global. Local methods, such as the **Kernighan-Lin algorithm**, focus on optimizing partitions based on local changes. In contrast, global approaches consider properties of the entire graph and do not depend on arbitrary initial partitions. One prevalent global approach is **spectral partitioning**, which involves deriving a partition from approximate eigenvectors of the graph's adjacency matrix, or **spectral clustering** that groups graph vertices using the eigendecomposition of the graph Laplacian matrix.

Consider a graph G = (V, E), where V denotes the set of n vertices and E the set of edges. For a (k,v) balanced partition problem, the objective is to partition G into k components of at most size v · (n/k), while minimizing the capacity of the edges between separate components [56]. Also, given G and an integer k > 1, partition V into k parts (subsets) $V_1, V_2, ..., V_k$ such that the parts are disjoint and have equal size, and the number of edges with endpoints in different parts is minimized.

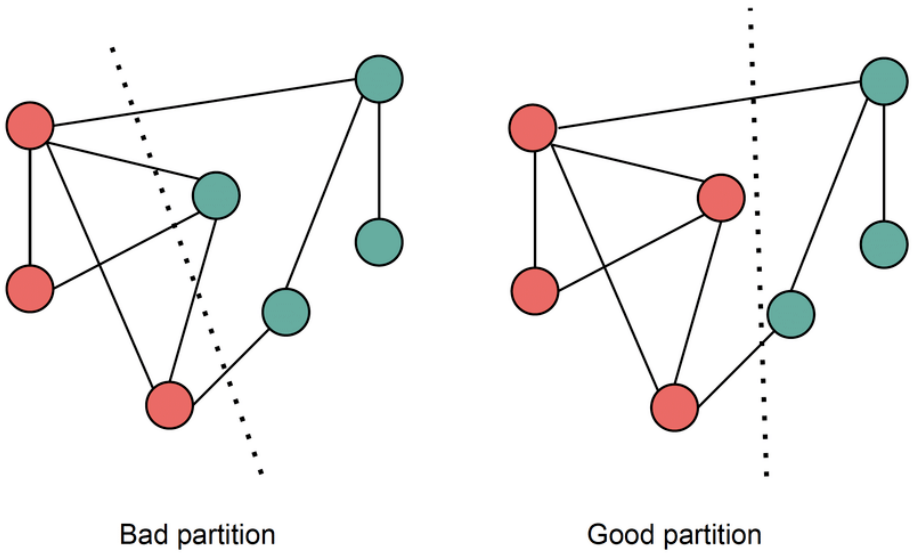


Figure 1.18: Graph partition example [16]

1.7.2 Clique Detection Problem

Identifying complete subgraphs (cliques) where every pair of vertices is connected by an edge. Common formulations of the clique problem include finding a maximum clique (a clique with the largest possible number of vertices), finding a maximum weight clique in a weighted graph, listing all maximal cliques (cliques that cannot be enlarged), and solving the decision problem of testing whether a graph contains a clique larger than a given size.

This is useful in social network analysis, bioinformatics, and finding tightly-knit groups within networks.

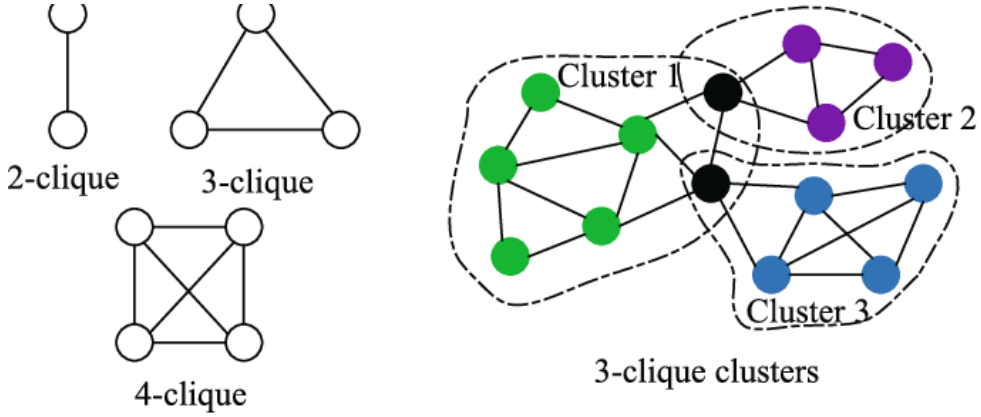


Figure 1.19: K-cliques and k-clique clusters [17]

Chapter 2

Overview of combinatorial optimization

2.1 Combinatorial Optimization

Combinatorial analysis involves the mathematical examination of organizing, grouping, ordering, or selecting discrete objects, often with a finite quantity. Traditionally, combinator ialists have addressed questions about the existence and counting of these arrangements [58].

Combinatorial optimization is a branch of mathematical optimization focused on identifying the best object from a finite collection of objects [59].deals with solving problems whose objective is to maximize (gain, performance,…: maximization problem) or minimize (loss, cost,…: minimization problem) an objective function under certain constraints [60].

In recent years, numerous studies have been published addressing both theoretical and practical optimization problems using various methods. Exact techniques aim to identify optimal solutions, while heuristic methods offer feasible solutions when exact approaches fall short of achieving optimality [61].

Optimization problems are broadly classified into two categories: exact and approximate. Exact algorithms provide precise solutions to problems, as their name implies. Approximate algorithms, on the other hand, may or may not yield exact solutions; instead, they offer approximate solutions. Approximate algorithms are further subdivided into two major categories: heuristic and metaheuristic algorithms. Heuristic algorithms encompass Local search, Divide and conquer, Branch-and-bound, Dynamic programming, and more. Metaheuristic algorithms include evolutionary algorithms, genetic algorithms, scatter search, simulated annealing, tabu search, guided local search, hill climbing, Iterated local search, and stochastic algorithms. [61].

Why optimization ?

- Countless applications across science, engineering, business, and economics.

- Optimization is integral to virtually every company.
- Every process can potentially be optimized
  - ⋆ Minimize: time, cost, risk…
  - ⋆ Maximize: profit, quality, efficiency … [62]

Optimization involves determining the conditions that yield either the maximum or minimum value of a function [63].

2.2 Solving optimization problems

Planning is seen as a structured, logical, and theory-driven approach to analyze and address planning and optimization challenges. The planning process involves multiple stages. [64]:

1. **Recognizing the problem**
2. **Defining the Problem**
   - ⋆ Identify the decision problem
   - ⋆ Identify internal / external objectives
   - ⋆ Determine the input (parameters)
   - ⋆ Determine Constraints
3. **Constructing a model for 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 [62]
   - **Simplified model, exact approach**

   - **Precise model, approximate approach**

4. **Solving the model**

   Once we've established a model for the original problem, it can be addressed using an algorithm, typically an optimization algorithm. An algorithm is a systematic set of well-defined instructions designed to complete a specific task. It begins with an initial state and concludes with a defined end-state. The objective of an algorithm is to identify a solution, which could be specific values for decision variables or a particular decision alternative, aiming to minimize or maximize the evaluation value [64].

5. **Validating the obtained solutions**

   Once optimal or near-optimal solutions are found, it is essential to evaluate them.

6. **Implementing Solutions**

Validated solutions must be put into action. This can occur in two ways: Firstly, a validated solution is implemented once. Secondly, the model is utilized and solved repeatedly.

2.3 Formal definition of optimization problem

An instance of a combinatorial optimization problem is a pair (X, f) where [62]:

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

In discrete (or combinatorial) optimization, our focus is on optimization problems in which the set of feasible solutions X for every instance $I = (X', f)$ is discrete. In other words, F is either finite or countably infinite.

In general, a combinatorial optimization problem $P(X, f)$ can be defined as follows [65]:

$$\begin{aligned} \text{Opt} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \geq 0, \quad i = 1, \ldots, m \\ & h_j(x) = 0, \quad j = 1, \ldots, p \\ & x \in X \end{aligned}$$

where Opt represents **Minimize** or **Maximize**. g_i **Inequality constraints** and h_i **Equality constraints** of the problem

The objective in addressing an optimization problem is to discover high-quality solutions. Ideally, these solutions should either pinpoint optimal solutions x^* (global optimum), near-optimal solutions $x \in X$, where the difference between $f(x)$ and $f(x^*)$ is minimal, or at the very least, locally optimal solutions [64].

2.3.1 Global optimum

A solution $x^* \in X$ is a global optimum if: [66]

⋆ $\forall x \in X$: $f(x^*) \leq f(x)$ (minimization problem)

⋆ $\forall x \in X$: $f(x^*) \geq f(x)$ (maximization problem)

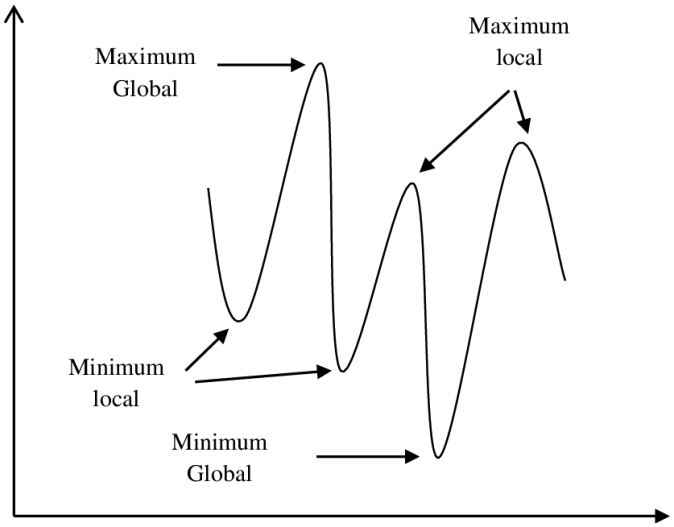


Figure 2.1: Global optimum and Local optimum [18]

2.3.2 Search Spaces

In optimization models, the search space X is implicitly determined by the definition of decision variables $x \in X$. Various essential aspects of search spaces include metrics used for assessing similarities between solutions in metric search spaces. Neighborhoods in a search space are established according to the metric utilized. Additionally, in combinatorial search spaces where a metric is defined, the concept of fitness landscape can be introduced. All these aspects of search spaces aid in the identification of locally and globally optimal solutions [64].

- X is solution space for an optimization problem
  - ⋆ X is empty that means no solution exists and the problem is too constrained
  - ⋆ X is non-empty that means one or more (even finite) optimal solution can exist (with the same value of f)

2.4 Classification of optimization problem

Most optimization models commonly used [67]:

- Mathematical programing
- Constraint programing

A constraint programming optimization model shares a similar structure with a mathematical programming model, consisting of decision variables, an objective function to maximize or minimize, and a set of constraints [68].

Mathematical program

Minimize / Maximize $f(x_1, ...x_n) \rightarrow$ objective function ($\mathbb{R} \rightarrow \mathbb{R}^n$) subject to [65]:

- **Inequality constraints:**

$$g_1(x_1, ...x_n) \geq b_1,$$
$$.........$$
$$g_m(x_1, ...x_n) \geq b_m,$$

- **Equality constraints:**

$$h_1(x_1, ...x_n) = c_1,$$
$$.........$$
$$h_m(x_1, ...x_m) = c_m,$$

There are many possible ways of classifying optimization problems since it is not well established and there is some confusion in literatur.Classification can carried out with respct to [69]

Tableau 2.1: Classification of optimization problem

Terms	**classified**
The existence of constraints	constrained vs unconstrained optimization
The function form(The type of objective function)	Linear vs nonlinear programming
The type of the value of the designe variable	Discrete vs continuous programming
The nature of parameters in the constraint Or the objective function	Stochastic vs deterministic programming
The number of decision variable	Finite vs non finite programming

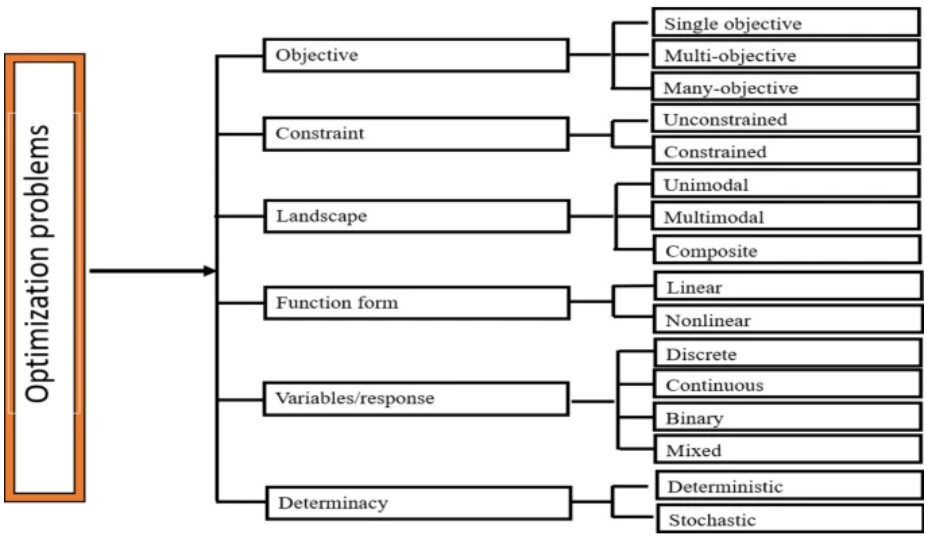


Figure 2.2: Classification of optimization problem [19]

2.5 Optimization vs decision problem

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

1. **Decision problem**
   A decision problem involves determining whether there is a solution to the problem, with the resolution limited to answering either ”yes” or ”no.” Consequently, it is not required to find the actual solution [70].

2. **Decision Problem Formalism**
   Formally, an instance of a decision problem can be represented as a binary string $s \in \{0,1\}^k$ where $|s| = k$ denotes the length of the binary string. We can denote the set of inputs corresponding to a ”yes” output as X which is a subset of all strings. For an input binary string s, we have... [70]:

   - $s \in X$ if and only if the output on s should be <yes>
   - $s \notin X$ if and only if the output on s should be <no>

   Consequently, a decision problem can be viewed as simply determining or deciding whether an input binary string is in set X. Thus, we can define the problem as this set X.

2.6 Algorithm and efficiency

An algorithm designed for an optimization problem offers a structured sequence of instructions that guides the computational process to address each given instance of the problem. Within the framework of our considerations, the Turing machine serves as the foundational computational model. When evaluating efficiency, our focus shifts from actual time measurements(such as minutes or seconds), to the count of fundamental operations involved. Various metrics can be employed to characterize an algorithm’s performance in terms of running time. In this context, we will use the worst-case running time, also known as computational complexity, as our chosen representation for assessing an algorithm’s overall efficiency.

The Different complexity Classes

a) **Class P:**

The class P contains all relatively easy decision problems, i.e. those that can be solved within a polynomial running time [71].This class contains many problems like Merge Sort,Calculating the greatest common divisor.
Features:

- The solution to P problems is easy to find.

- P often refers to a class of computational problems that are both solvable and manageable. ”Manageable” implies that these problems can be solved both in theory and in practice. However, problems that are solvable in theory but not in practice are termed intractable.

b) **Class NP:**

The class NP is defined as the class of decision problems X that can be solved by non-deterministic algorithm within a polynomial running time. [72].This class contains many problems that one would like to be able to solve effectively:Hamiltonian Path Problem,Graph coloring.

Features:

- Solutions within the NP class are challenging to find due to their resolution by a non-deterministic machine, but once found, they are easy to verify.
- NP problems can be verified by a Turing machine in polynomial time.

c) **Class NP-Hard:**

An NP-hard problem is at least as challenging as the most difficult problem in NP, and it’s a category of problems to which every NP problem can be reduced. [73].example of problems in Np-hard are: No Hamiltonian cycle.

Features:

- All NP-hard problems are not in NP.
- Verifying solutions for NP-hard problems is time-consuming. Therefore, if a solution to an NP-hard problem is provided, determining its correctness requires significant time.
- A problem A is classified as NP-hard if there exists a polynomial-time reduction from every problem L in NP to A.

d) **Class NP-complet:**

A problem is considered NP-complete if it belongs to both NP and is NP-hard. NP-complete problems represent the most challenging problems within NP [74].Some example problems include:Hamiltonian Cycle,Vertex cover.

Features:

- NP-complete problems are unique in that any problem within the NP class can be transformed or reduced to an NP-complete problem in polynomial time.
- If an NP-complete problem could be solved in polynomial time, then any problem in NP could also be solved in polynomial time.

Complexity Class	Characteristic feature
P	Easily solvable in polynomial time
NP	Yes, answers can be checked in polynomial time.
NP-hard	All NP-hard problems are not in NP and it takes a long time to check them.
NP-complete	A problem that is NP and NP-hard is NP-complete.

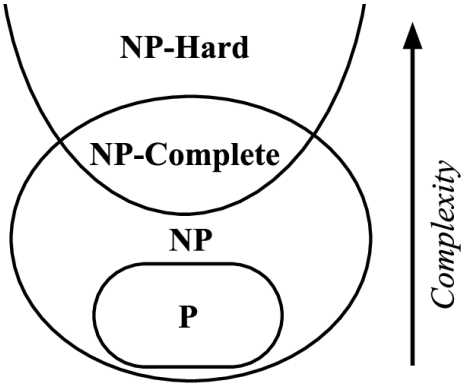


Figure 2.3: The relationships among complexity classes of problem [20]

2.7 Optimization methods

In general, optimization algorithms can be categorized into two fundamental classes: deterministic and probabilistic algorithms.

2.7.1 Deterministic algorithms

Deterministic algorithms are primarily employed when a direct correlation between the attributes of potential solutions and their effectiveness for a given problem is evident. In such cases, the search space can be efficiently navigated, often utilizing strategies like divide and conquer. However, if the relationship between a solution candidate and its ”fitness” is less clear or complex, or if the dimensionality of the search space is extensive, solving the problem deterministically becomes challenging. Attempting to do so may necessitate exhaustive enumeration of the search space, which is impractical even for relatively modest problems. [64]

2.7.2 Probabilistic algorithms

Probabilistic algorithms model or explore a problem space using a probabilistic framework for candidate solutions. Many metaheuristics and computational intelligence algorithms fall into this category, distinguishing themselves from deterministic algorithms by explicitly incorporating the tools of probability in problem-solving. While these algorithms sacrifice the guarantee

of achieving the optimal solution, they often offer shorter runtimes. This doesn't imply that the results obtained are incorrect; they may simply not be the global optimum. However, a solution that is slightly suboptimal is preferable to one that would require an astronomically long time to find [64].

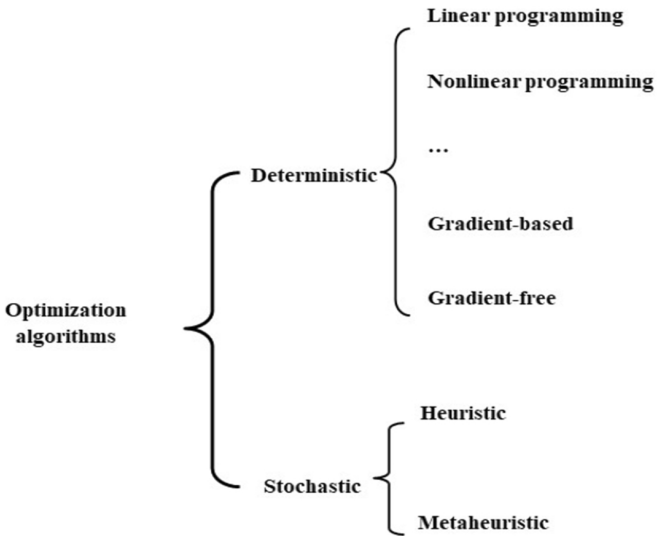


Figure 2.4: Classification of algorithms [19]

2.7.3 Solving Combinatorial Optimization Problems

There are two categories of algorithms commonly used to solve combinatorial optimization problems: exact and approximate algorithms.

2.7.3.1 Exact Methods

Exact methods aim to ascertain the optimal solution definitively by thoroughly exploring the entire search space, either explicitly or implicitly. While they offer the advantage of guaranteeing the optimal solution, their computational requirements can become prohibitively excessive, particularly for larger problems due to combinatorial explosion and the number of objectives to optimize. As a result, the practical applicability of these methods is often limited to smaller-sized problems. Examples of such generic methods include Branch & Bound, Dynamic Programming, Constraint Programming, and the Simplex method … [75].

In the case of NP-hard problems, exact algorithms need, in the worst case, exponential time to find the optimum. For most NP-hard problems the performance of exact algorithms is not satisfy factory [76].

- **Branch and Bound:**

  The Branch and Bound Algorithm is a technique employed in combinatorial optimization problems to methodically search for the optimal solution. It operates by subdividing

the problem into smaller subproblems, or branches, and subsequently eliminating certain branches based on bounds on the optimal solution. This iterative process persists until either the best solution is discovered or all branches have been explored. Branch and Bound is frequently applied to problems such as the traveling salesman and job scheduling. [77].

- **Dynamic Programming**

  Dynamic Programming is a problem-solving technique that tackles complex problems by decomposing them into smaller, more manageable subproblems. By solving each subproblem just once and storing the results, it minimizes redundant computations, thereby offering more efficient solutions for a diverse array of problems [78].

2.7.3.2 Approximate Methods

These methods are used for problems where no algorithms are known resolution in polynomial time and for which one seeks to obtain a ”good” solution, without any guarantee that it will be the best. So, they are very useful to be able to approach larger size issues. They bring together heuristics specific to a Particular POC and metaheuristics. The former is not very reusable (the methods constructive, greedy, . . .). On the other hand, metaheuristics are more general and are independent of the processed POCs [75]. In this work, we are exclusively interested in metaheuristic.

I. **Heuristics**:

   For some problems, the algorithms are too complex to obtain a result in a reasonable time, even if one could use a power of phenomenal calculation. We are therefore led to seek a solution as close as possible to an optimal solution by proceeding by successive tests. Since not all combinations can be tried, certain strategic choices must be made. These choices, generally very dependent on the problem treated, constitute what is called a heuristic. The goal of a heuristic is not to try all the possible combinations before finding the one which answers the problem, in order to find a suitable approximate solution (which can be exact in some cases) within a reasonable time. In order to resolve problems and decision-making, heuristics nevertheless find their place in the algorithms that require the exploration of a large number of cases, because these allow us to reduce their average complexity by first examining the cases that are most likely to give the answer [75].

II. **Metaheuristics:**

   Metaheuristics have grown considerably since their appearance in the 1970s. They are presented by Osman and Laporte (1996) as being approximation methods designed to many complex optimization problems that could not be solved effectively by heuristics and methods of classical optimization. These same authors formally define the notion

of metaheuristic as an iterative process that guides a subordinate heuristic in intelligently combining different concepts to explore and exploit the research space, and who uses learning strategies to structure information in order to find efficient solutions as close as possible to the optimal solution. The development of metaheuristics is part of a sustained effort invested in the field of combinatorial optimization [75].

The main aim is to ensure a balance between:
- Exploration of search space
- Exploitation of the information gathered during the previous search

we can classify metaheuristics into two large families:

1. Metaheuristics based on a single solution:
   - Local search LS
   - Simulated annealing SA
   - Tabu search TS
   - Hill Climbing HC

2. Metaheuristics with a population of solutions:
   - Evolutionary algorithms :
     - Genetic algorithm GA
     - Genetic programming GP
     - Evolution strategies ES
     - Evolution programming EP
   - Swarm intelligence algorithm:
     - Particle swarm optimization PSO
     - Ant colony optimization ACO
     - Artificial Bee Colony ABC
     - Bat Algorithm BA

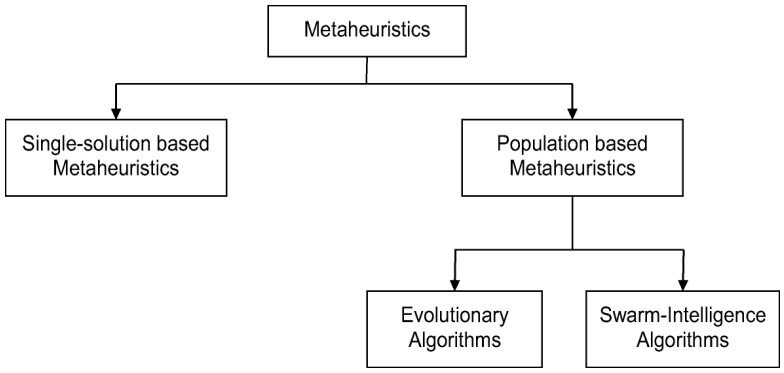


Figure 2.5: Metaheuristic taxonomy

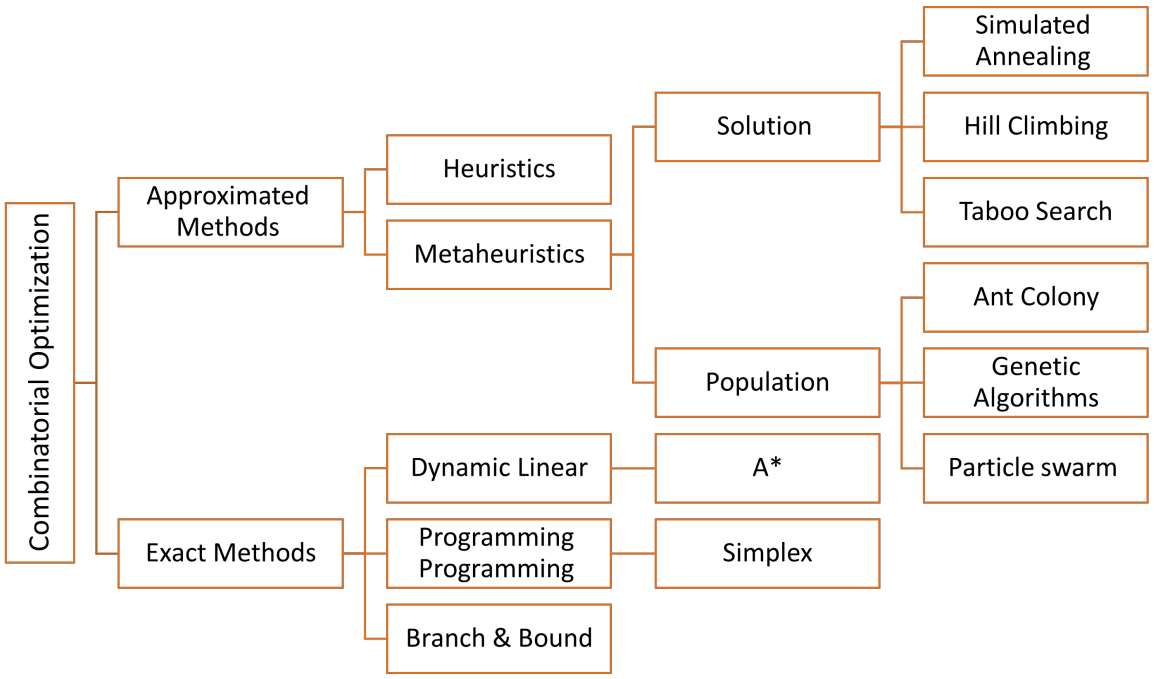


Figure 2.6: Classification Of Combinatorial Optimization Methods

Chapter 3

Community detection

3.1 Introduction

Numerous complex systems across various domains like biology, computer science, linguistics, and commerce can be abstractly represented as networks. A prevalent characteristic of many networks is the presence of regions with higher density of connections compared to others. These regions are commonly known as communities, representing groups of nodes that exhibit stronger connections among themselves than with other nodes in the network. Therefore, the objective of community detection is to categorize network members into such groups [79].

To detect communities in a given network, we need an easy and practical representation such as a graph.

3.2 Communities

The notion of communities in graphs has no formal definition. However, the existence of areas that are more densely connected than others is the result of the presence of graph structures whose nodes have grouped together into communities due to their similarity or common interests [80].

Here we give two definitions of communities, one semantic and the other structural [42, 81]:

- **Semantic definition**: A community comprises nodes that share similar interests or characteristics or have the same profile.
- **Structural definition**: A community consists of nodes that exhibit strong connections among themselves while having weaker connections to nodes outside the community in the graph.

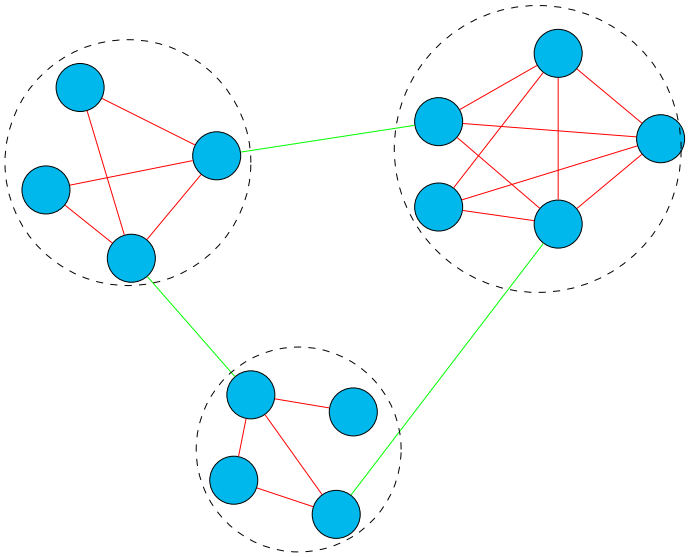


Figure 3.1: A simple graph with three communities

3.2.1 Community structure

The group of communities discovered in the network is called the community structure It is represented as : $C = C_1, C_2, C_3, ..., C_k$ where C it is the community structure, $C_1, C_2, ..., C_k$ represent the communities [43].

For example, in Figure. 3.2 the communities are : C_1 = (1,,2,3,4,5,6,7,8,9), C_2= (10,11,12,13,14,15), C_3 = (16,17,18) and the community structure is $C = C_1, C_2, C_3$.

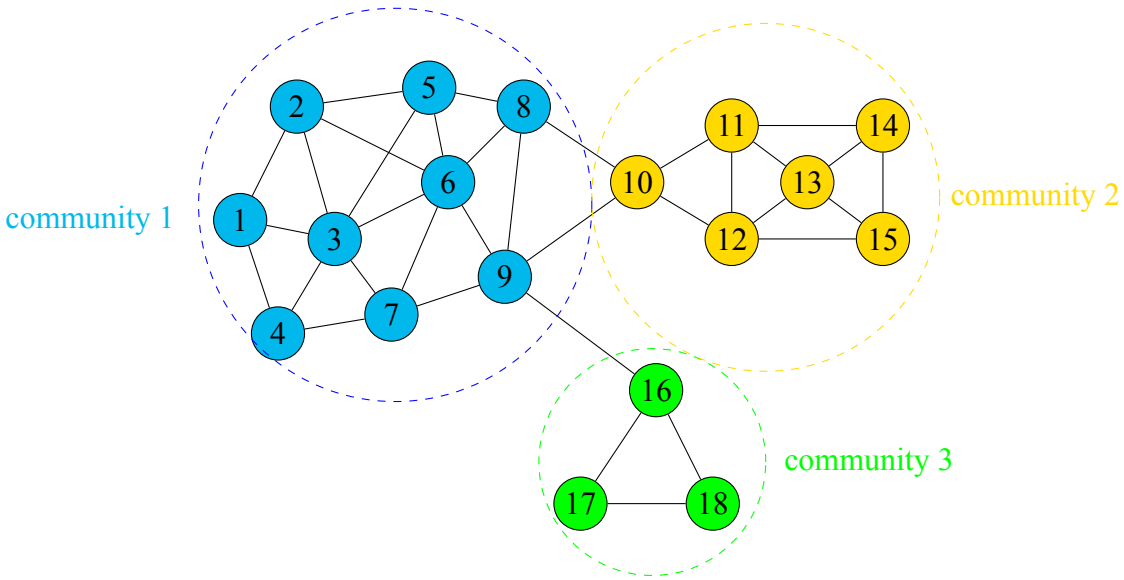


Figure 3.2: The community structure of an example network

The community structure where some nodes overlap is known as the overlapping community structure (see Figure. 3.3). The community structure where all nodes are non-overlapping is known as a disjoint community structure. [81].

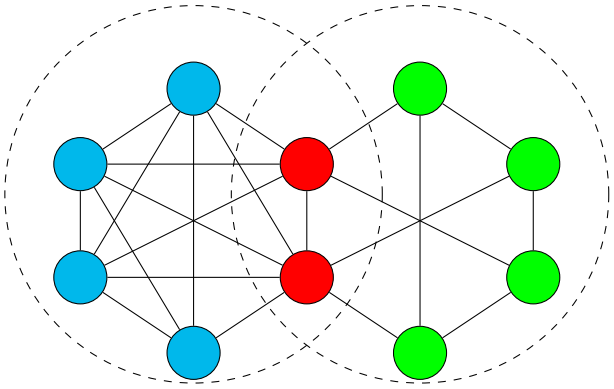


Figure 3.3: A simple network with two overlapping communities

Community detection involves identifying network nodes that are more densely interconnected than others, thereby forming cohesive groups. These groups are then constructed based on a similarity measure, typically reflecting shared interests or characteristics, to categorize users into related classes [82].

3.3 Community detection

The search for community partitions is one of the main problems associated with network analysis. The methods for solving it have been the subject of a great deal of work, since the seminal article by Girvan and Newman.

Community detection is a network analysis task aimed at identifying groups of closely related entities in a network. The aim is to find subgroups of entities that have stronger links to each other than to entities outside the group [83].

3.3.1 Objectives of community detection

The goals of this concept of communities include the following objectives [84]:

- The aim of community detection is to uncover novel relationships and extract hidden properties within a network represented as a graph. This process aids in comprehending the underlying structure of the network, revealing valuable information, and facilitating various analytical tasks.
- Enhancing network comprehension: By identifying cohesive groups of nodes, this process facilitates a deeper understanding of the network's organization and dynamics.
- Provide a summary of the network structure.

- Identifying key players: Understanding the central nodes or influential actors within communities helps in pinpointing critical elements that impact the network's stability and functionality.
- Revealing new insights: Community detection uncovers novel insights and properties within the network, offering valuable information for various analytical tasks and decision-making processes.
- Study the similarity among individuals within the same community to measure the extent of interaction, and subsequently quantify the strength of their relationships.
- Understanding collective behavior: By studying community structures, researchers gain insights into collective behavior, preferences, and trends exhibited by groups of individuals.
- Understanding which way people lean politically and what they like or think about products on the market.
- Analyzing interactions: Community detection allows for the analysis of interactions among individuals within the same group, providing insights into the strength and nature of their relationships.
- Implementation of a marketing strategy (recommendation systems): based on knowledge of the community as a whole, we can identify the different profiles (or common interests grouping together members), and then we can distribute (advertising, personalized recommendations, etc.) precise information to a well-known set of users.
- Solutions for minimizing / maximizing diffusion.
- Enhancing network efficiency: By optimizing community structures, network efficiency can be improved, leading to better communication, resource allocation, and overall performance.
- Facilitating targeted interventions: Knowledge of community structures enables targeted interventions and strategies tailored to specific groups, such as marketing campaigns or security measures.

3.3.2 Applications of community detection

There are many applications for community detection. Here are just a few:

a) **Application in biological and public health**
Detection of communities in biological networks is of great significance, such as protein networks, food webs, metabolic networks, etc.... In protein networks, it has been applied

to detect protein complexes [85].
In the health domain, community detection is commonly employed to uncover the dynamics of specific groups vulnerable to epidemic diseases. It is also utilized for identifying diseases such as cancer and tumor types. Moreover, community detection plays a role in organ detection [86].

b) **Application in criminology**
Community detection is employed to identify groups of criminal users, which may consist of real individuals or bot accounts. These groups may promote or disseminate criminal ideas or engage in activities resembling terrorism. By applying community detection, these criminal networks can be identified, followed by manual analysis to further understand their dynamics and behaviors [86]

c) **Application in Social Network Analysis**:
Community detection serves as a valuable tool for comprehending communities at a networking level and correlating them with real-life relationships. For instance, applying community detection to social networks like Facebook, Twitter, and LinkedIn offers insights into their structures. Social Network Analysis stands out as one of the most commonly utilized methods for community detection in this domain [87].

d) **Application in politics**
In politics, community detection serves to observe the impact of political ideologies or specific politicians on social groups. Furthermore, it can be tailored to monitor the evolution of this influence over time. This influence is often facilitated by influence or astroturfer bots, which aim to create a false impression among genuine grassroots supporters to endorse a policy, individual, or product campaign [88].

e) **Application in Recommendation Systems**
Recommendation systems play a crucial role in our daily lives, assisting us when we wish to purchase a book online, watch a video, or listen to music on social media platforms. These systems aim to suggest items that align with our preferences and interests. Community detection, on the other hand, involves grouping individuals with similar mindsets. While numerous studies in the literature incorporate community detection into recommendation systems, the primary objective remains to enhance the accuracy and relevance of recommendations provided to users [89, 90].

f) **Applications Scientific and academic**
Community detection algorithms are useful in scientific research. Their usefulness lies in their ability to classify authors, their publications, the years and places of publication, etc…These algorithms can predict new relationships between authors (scientific collaboration) and can propose new papers to authors according to their profile. This system

can be reduced to the case of a simple library where we can propose books to students according to their specialty, analyze the similarity between books and between students, etc… [91].

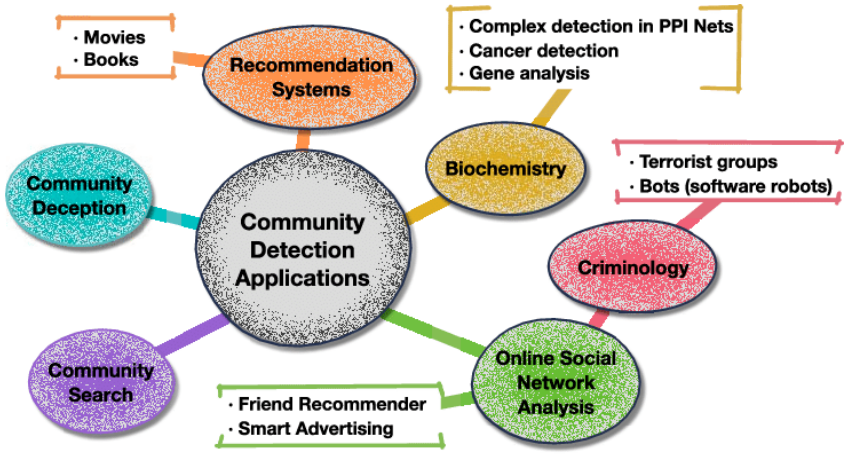


Figure 3.4: Some of applications of community detection [21]

3.3.3 Classifications of community detection methods

Numerous classifications of community detection methods have been documented, predominantly organized around the types of algorithms used and their underlying principles. Santo Fortunato [80] conducted a comprehensive investigation, categorizing these methods into eight distinct groups,as follows:

1. **Traditional methods:** The approach involves the optimal division of graphs representing networks into k partitions or ”clusters,” with k predetermined. One notable algorithm for this purpose is the Kernighan-Lin algorithm [92]. Initially, the objective was to find partitions of similar sizes. However, this strict constraint proved challenging to meet in practical scenarios. Consequently, the approach was relaxed to focus on identifying communities without specifying exact [93]. Hierarchical partitioning is then guided by a similarity function, ensuring that ”clusters” consist of nodes with significant similarities.

2. **Divisive algorithms:** These methods revolve around identifying and removing a characteristic of inter-community links, leading to the disconnection of the graph and the formation of cohesive components representing communities. The Girvan and Newman algorithm is widely recognized as one of the most prominent approaches in this category. [94].

3. **Modularity-based methods** Modularity, introduced by Girvan and Newman, serves as a quality measure utilized by numerous algorithms through optimization or adjustment processes [95]. However, identifying the optimal partition based on modularity is proven to be an NP-complete problem. Consequently, alternative approaches like the Louvain algorithm rely on greedy techniques to offer satisfactory solutions within reasonable computation time.

4. **Spectral algorithms** These algorithms leverage the concept of a spectrum to delineate the proximity between nodes. Eigenvectors, serving as temporal propagators in the random walk process within the social network graph, are linked to the lowest eigenvalues, characterizing groups with significant internal coherence [96]. Typically, the Laplacian matrix is employed as the similarity matrix in this context.

5. **Dynamic algorithms** Simulate dynamic processes where particles influence each other. For example, particles close to each other tend to share the same state. Among the processes applied to Social Network graphs we cite Spin-Spin and synchronization, where the system progressively unifies all its elements to the same state [97].

6. **Methods based on statistical inference** Similar to Bayesian inference, which encompasses generative models and block modeling, these methods operate under the assumption that the graph is generated based on a model that includes node memberships to communities as parameters. The objective is to infer the parameters that would most likely produce the observed data [42].

7. **Methods to find overlapping communities** Obviously, a node can belong to more than one group or community - this is the characteristic of community overlap. The first method to effectively take into account overlap was proposed by Palla in 2005 [98], and other approaches have subsequently been proposed.

8. **Multiresolution methods and cluster hierarchy** The application of the multi-resolution paradigm to community detection seeks to incorporate a scaling factor that enables the detection scale to be determined directly and the characteristic community size to be determined indirectly [99]

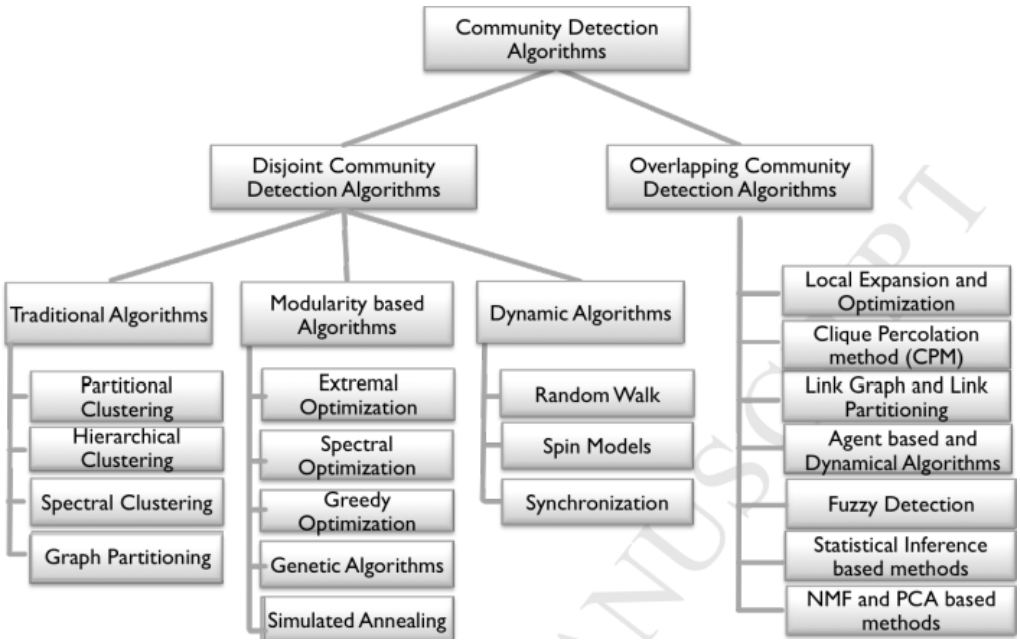


Figure 3.5: Classification breakdown of algorithms for community detection [22].

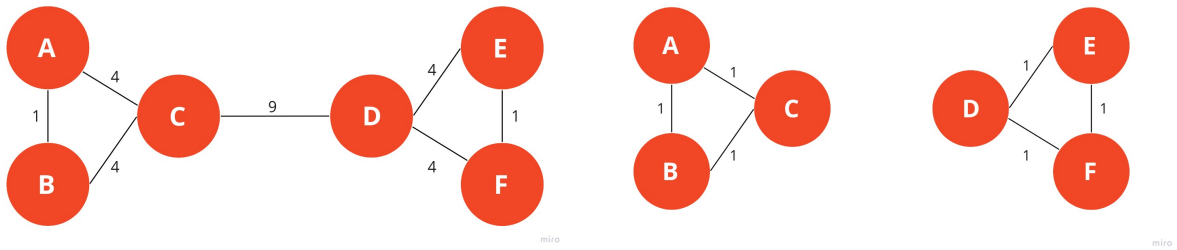
3.3.4 Community detection algorithms

3.3.4.1 Newman-Girvan algorithm

This is the most classic separative method, which introduced a centrality measure called Edge-Betweenness Centrality to partition a graph. This centrality measure is defined as the number of shortest paths between two nodes that pass through an edge. This algorithm is particularly intuitive [81].

The idea behind this algorithm is as follows: if a link is frequently found on the shortest paths between graph nodes, then it is not within a given community, but connects distant portions of the graph (distinct communities) [42].

The first step is to calculate this edge betweenness for all the edges in the graph, then remove the edge with the highest betweenness. This process is iterated until the last edge is removed. In a second phase, starting from the edge-free graph, the edges are reintroduced in reverse order, providing a very fine-grained hierarchy of the network, since by adding an edge linking two communities, we add a hierarchical level, the two communities being grouped together in a bigger community [100].



(a) **(b)**

Figure 3.6: Exemple of Girvan-Newman algorithm

The Girvan-Newman algorithm would eliminate the edge connecting nodes C and D (Figure. 3.6b) because it possesses the highest strength. Intuitively, this indicates that the edge lies between communities. Following the removal of an edge, the betweenness centrality must be recalculated for all remaining edges. In this scenario, we have reached a state where every remaining edge shows the same betweenness centrality [101].

3.3.4.2 Louvain algorithm

The Louvain community detection algorithm, initially introduced in 2008, serves as a rapid method for uncovering communities within extensive networks. This technique relies on modularity, aiming to enhance the disparity between the observed number of edges within a community and the anticipated number of edges. However, optimizing modularity within a network is

NP-hard, necessitating the utilization of heuristics [102]. The Louvain algorithm is structured into two phases that iteratively repeat:

- Local moving of nodes
- Aggregation of the network

it starts with the assumption that every node is a community, then groups every two adjacent nodes into a single community by maximizing modularity. It comprises two phases. First, it searches for ”small” communities by optimizing modularity locally (separately, at the level of each community). Secondly, it groups the nodes of the same community and builds a new network whose nodes are the communities, with the sum of the weights of the edges between the two communities as weights. These two phases produce a new hierarchical level of community partitioning, and the algorithm stops when none of the mergers from the first phase improves modularity any further [84, 103].

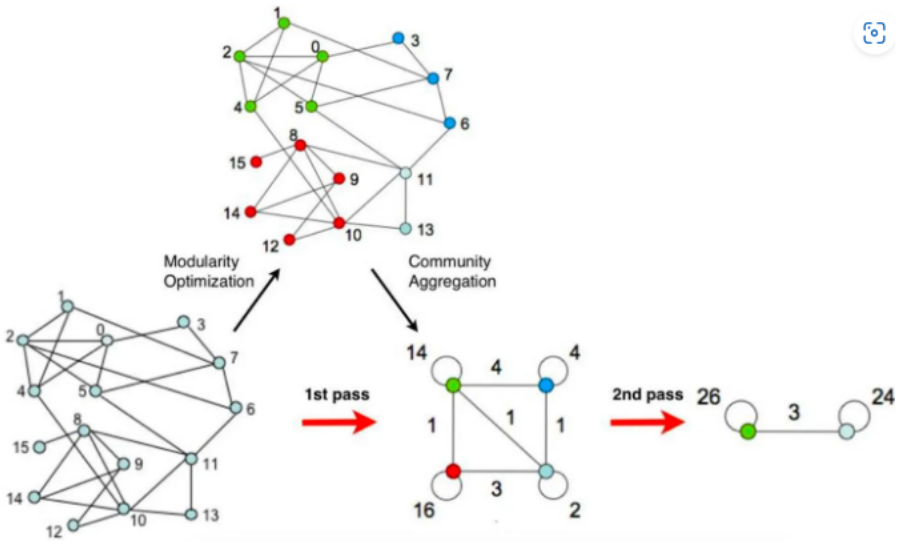


Figure 3.7: Example of two phases in Louvian Algorithm [23].

The Louvain community detection algorithm reveals communities during the process, making it popular due to its easy implementation and fast execution. Nonetheless, a significant drawback lies in its requirement for storing the entire network in main memory.

3.3.4.3 The Label Propagation algorithm (LPA)

The main idea of this algorithm is that each vertex (X) in the network is assigned a unique label (its own community) [104], then (X) determines its community based on the labels of its neighbors. (X) belongs to the community containing the largest number of neighboring nodes, an iterative process is executed so that the groups of connected vertices can reach consensus on

a label giving rise to a community, at the end of the propagation process, nodes with the same labels are grouped together into a single community [105].

The algorithm works as follows [106]:

- Every node is initialized with a unique community label (an identifier).
- These labels propagate through the network.
- At every iteration of propagation, each node updates its label to the one that the maximum numbers of its neighbours belongs to. Ties are broken arbitrarily but deterministically.
- LPA reaches convergence when each node has the majority label of its neighbours.
- LPA stops if either convergence, or the user-defined maximum number of iterations is achieved.

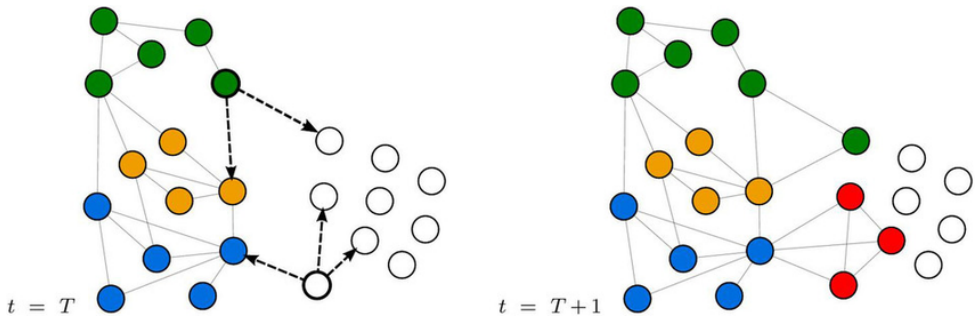


Figure 3.8: Two representative examples of label propagation [24].

3.3.5 Measures to evaluate the quality of community structures

Currently, there are a number of measures available to evaluate the efficiency of community detection algorithms.Some of These metrics include :

3.3.5.1 Modularity (Q)

This is one of the metrics frequently used to measure the quality of community detection in networks, and was proposed by Girven and Newman in 2004. Networks with high modularity have dense connections between nodes within modules, but sparse connections between nodes in different modules. Modularity (Q) is defined by [107, 108]:

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i \cdot k_j}{2m} \right) \delta(c_i, c_j) \tag{3.1}$$

where :
A_{ij}: Represents the network's adjacency matrix.

c_i: Represents the community to which node i is assigned.
c_j: Represents the community to which node j is assigned.
k_i: Represents the degree of node i.
k_j: Represents the degree of node j.
m: Is the number of edges in a network.
Thus, the function $\delta(c_i, c_j)$ can be defined as follows:

$$\delta(c_i, c_j) = \begin{cases} 1, & \text{if nodes } i \text{ and } j \text{ are in the same community,} \\ 0, & \text{otherwise.} \end{cases}$$

The value of Q ranges from -1 to +1. The closer the value is to 1, the greater the strength of the community structure in the network, and the better the quality of community detection [108].

3.3.5.2 Normalized mutual information (NMI)

It is a similarity measure that estimates the similarity between two partitions A and B; whereA represents the real partition of the network and B the partition detected by experimental community detection algorithms. It is based on information theory [109].

For two partitions A and B of a network, the value of NMI is calculated by equation [107]:

$$\text{NMI}(A, B) = \frac{-2 \sum_{i=1}^{C_A} \sum_{j=1}^{C_B} N_{ij} \log\left(\frac{N_{ij} N}{N_i N_j}\right)}{\sum_{i=1}^{C_A} N_i \log\left(\frac{N_i}{N}\right) + \sum_{j=1}^{C_B} N_j \log\left(\frac{N_j}{N}\right)} \tag{3.2}$$

where:
A: represents the actual partition of the network.
B: the partition discovered by the community detection algorithms.
C_A: represents the number of communities in partition A.
C_B: designates the number of communities in partition B.
N: represents the total number of of vertices in the network.
N_{ij}: represents the number of identical of vertices in community i in partition A and the j^{th} community in partition B.
N_i: is the number of vertices in real community i (the sum of row i of matrix N_{ij}).
N_j: is the number of vertices in calculated community j (the sum of column j).

The NMI value can vary between 0 and 1. The closer the NMI value is to 1, the more similar the two partitions are.

In other words, when two partitions A and B are completely different, then $\text{NMI}(A, B) = 0$.

If NMI takes its maximum value, which is 1, then partition A corresponds exactly to partition B.

3.4 Communities in real-world networks

In the literature, a collection of real data sets exist in the form of graphs, which are not designed at random but are based on real social properties. In other words, these networks graphically represent human behaviors that can be exploited and analyzed. There are many real-life networks, but here are a few of the most popular:

3.4.1 Zachary's karate club network:

The Zachary karate club network [110], illustrated in Figure Figure. 3.9, is a well-known network regularly used as a reference for testing community detection algorithms.This network comprises two communities.

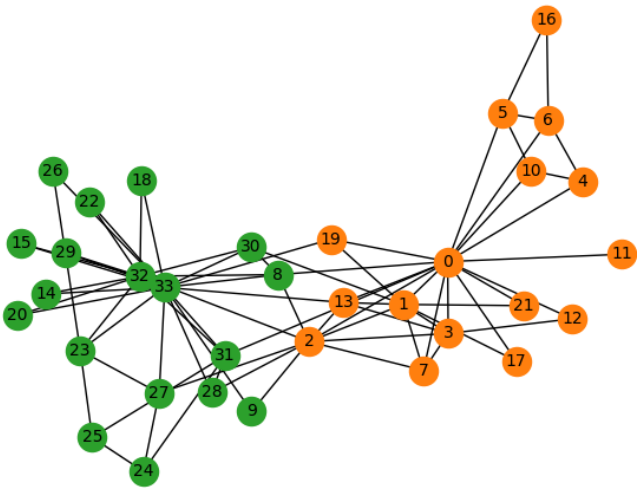


Figure 3.9: Zachary's karate club network

The graph comprises 34 vertices and 78 edges, representing members of a karate club in the United States over three years. The edges denote interactions observed outside club activities. A conflict between the club president and the instructor resulted in the club splitting into two factions, supporting either the instructor or the president (denoted by squares and circles). Observing Figure. 3.9, two distinct clusters are evident: one centered around vertices 33 and 34 (with 34 representing the president), and the other around vertex 1 (the instructor). [110].

3.4.2 Dolphin network

The Dolphin Network [111] was developed by authors who observed the lifestyle of dolphins that have long lived in the magical fjords of New Zealand. It is a network of social relationships between dolphins. The frequent exchanges between dolphins constitute a connection between different dolphins, consisting of 62 nodes and 159 edges (see Figure. 3.10). The nodes represent the dolphins and the edges represent the interactions (the frequent associations between each two dolphins) between the dolphins. This network comprises two communities, male and female.

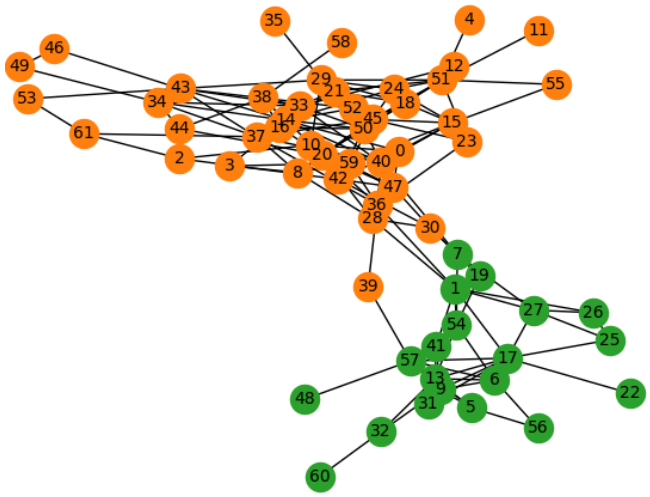


Figure 3.10: Dolphin network

3.4.3 American football network

Another example of a real network is the American soccer game network [112]. It represents the schedule of matches between American soccer teams during the year 2000. This network is made up of twelve communities (see Figure. 3.11), 115 nodes and 613 links. Each node represented a football team, while each edge represented the relationship between two teams.

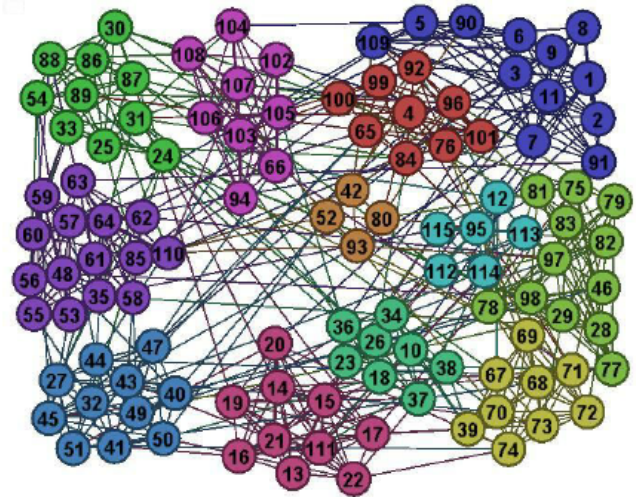


Figure 3.11: American football network [25]

3.4.4 American Politics Books Network

This dataset is the Amazon co-purchasing network with 105 books on American politics. There are 441 edges (see Figure. 3.12). The nodes are books and the edges represent the co-purchase of books by the same buyers. The network has 3 communities: Democrats, Republicans and neutrals [94, 113].

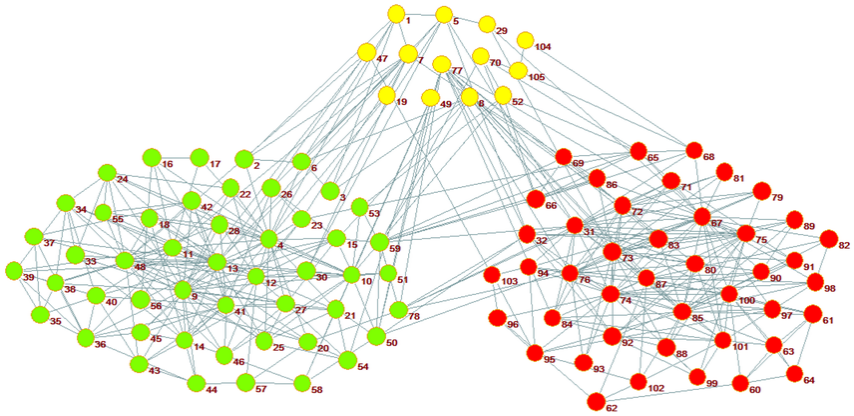


Figure 3.12: American Politics Books Network [26]

Chapter 4

Community Detection Algorithm Based on Discrete Particle Swarm Optimization

4.1 Introduction

Finding groups in complex networks is important for understanding how things like social networks, biological systems, and transportation systems are organized. But current methods for doing this can be really complicated and slow and need prior knowledge about the size of communities and the number of communities, especially when dealing with big networks with lots of different parts.So, we need new ways to find these groups that work well and are fast. In our research, we came up with a simple and efficient method using something called discrete particle swarm optimization (DPSO) algorithms.

In this chapter we present fundamental Principles of Particle Swarm Optimization Algorithm. Then See Improved discrete particle swarm optimization algorithm have been suggested by [114] and how it work. Lastly Introducing a modification to the community detection method based on discrete particle swarm optimization.

4.2 Objectives

This research aims to present a fresh methodology for community detection in intricate networks through DPSO algorithms. Our specific goals are:

- Formulate a discrete particle swarm optimization algorithm customized for community detection in complex networks.
- improve algorithm can detect community structures more efficiently without prior knowledge about the size of communities and the number of communities.
- Design a simple method that is easy to implement and applicable to complex networks.

- Evaluate the efficacy of the devised approach in precisely delineating communities across diverse network categories.
- Compare the performance of our method with existing community detection techniques concerning both the accuracy of community detection and computational efficiency.
- Validate the scalability of our method and its adeptness in managing complex networks show heterogeneous structures.

4.3 Related work

In recent years, numerous community detection methods have been developed, each employing different strategies. One significant branch of these algorithms is the optimization-based approach, which frames community detection as a combinatorial optimization problem. These methods identify community structures by optimizing a predefined evaluation criterion that reflects a specific property of the community, such as modularity.

A kind of classical approach to community detection is to transform the problem into an optimization problem and identify the community structure by optimizing specific objective functions, such as modularity and module density [115]. There are three notable optimization algorithms used for this purpose: the Kernighan-Lin algorithm [116], the genetic algorithm (GA) [117, 118], and the particle swarm optimization (PSO) algorithm [119, 120].

The Kernighan-Lin algorithm detects communities by maximizing a gain function [116]. However, it has the limitation of only being able to divide the network into two communities at a time, rather than identifying all communities simultaneously. Additionally, it requires prior knowledge about the size of the communities, which restricts its applicability in practice.

Genetic algorithms are commonly employed for community detection [117, 118]. Firat et al. use a random walk distance metric in a genetic algorithm for clustering [117]. However, its applicability is limited due to the need for prior knowledge about the number of clusters. Pizzuti introduces a multiobjective genetic algorithm (MOGA-Net), which optimizes two objective functions: community score and community fitness [118]. Despite its potential, MOGA-Net faces several challenges, including an excessive number of parameters, slow convergence rate, and low success rate. Additionally, the parameter used to control community size is difficult to define.

In [119], Chen used similarity-based clustering to identify core areas in the network, followed by a modified PSO algorithm to optimize modularity in a newly constructed weighted

network. However, the treatment of isolated nodes, which impact the PSO algorithm's performance, was not addressed. Duan et al. proposed a Web community detection model using PSO [121], which efficiently detects communities without prior information, but it is limited to finding only two communities at a time and has stability issues due to network feature neglect during initialization. A discrete PSO algorithm was used in [120] for community detection, but it could only identify two communities per iteration, making it unsuitable for multi-community networks and lacking problem-specific optimizations. Cai et al. presented an effective discrete PSO paradigm for signed network community detection, but it did not incorporate local search strategies to enhance algorithm performance [122].

The particle swarm optimization (PSO) algorithm is characterized by its few parameters, fast convergence, and ease of implementation. It has been widely applied to the NP-hard problem of community detection in complex networks [119, 120]. One of the key advantages of the PSO algorithm is that it does not require prior knowledge about the size or number of communities. Therefore, in this recherche, we choose the PSO algorithm to detect community structures.

4.4 Fundamental Principles of PSO Algorithm

The Particle Swarm Optimization (PSO) algorithm was introduced by Eberhart and Kennedy in 1995, drawing inspiration from the collective foraging behavior of bird flocks [123]. In PSO, a group of individuals is deployed within the search space of a problem or function. Each individual assesses the fitness function at its current position and adjusts its movement based on a combination of its current position, its best position found thus far, and the best position of the entire population. Iterations proceed until convergence towards an optimal solution of the fitness landscape [124].

Consider a population of m individuals in a D-dimensional search space. Each individual's position is represented by a D-dimensional vector, $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$. The fitness function evaluates the suitability of an individual's current position, providing a fitness value for each individual. This fitness value informs the individual's position quality. The best position an individual has ever attained is denoted as $P_i = (p_{i1}, p_{i2}, \ldots, p_{iD})$, where $i = 1, 2, \ldots, m$. The optimal position discovered by any individual within the population is represented as $P_g = (p_{g1}, p_{g2}, \ldots, p_{gD})$.

Shi and Eberhart incorporated a novel parameter, the inertia weight, into the initial particle swarm optimization algorithm [125]. Studies have indicated that a larger inertia weight ω fosters global exploration, whereas a smaller value promotes local exploration. The update equations for velocity and position are as follows:

$$v_{id} = \omega * v_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * rand() * (p_{gd} - x_{id}) \tag{4.1}$$

$$x_{id} = x_{id} + v_{id} \tag{4.2}$$

$$\omega = \omega_{\text{max}} - \frac{\omega_{\text{max}} - \omega_{\text{min}}}{\text{TotalIter}} \times \text{CurIter} \tag{4.3}$$

Here, **i** ranges from 1 to m, and **d** ranges from 1 to D. **c1** and **c2** represent constants, ω denotes the inertia weight, and ω_{max} and ω_{min} signify the maximum and minimum values of ω. **CurIter** denotes the current iteration count, while **TotalIter** represents the maximum number of iterations.

4.5 The proposed algorithm by authors

PSO algorithm Originally developed for continuous function optimization,but many problems are discrete in practical applications, as seen in community detection problems. In these cases, position values must be constrained to integer spaces, prompting adaptations of the PSO algorithm for such scenarios.

So **Cen Cao**, **Qingjian Ni** and **Yuqing Zhai** proposed 2 algorithm **ISDPSO** and **IDPSO-RO** [114] to deal with discrete problem like community detection .

4.5.1 Simple Discrete Particle Swarm Optimization

The Improved Simple Discrete Particle Swarm Optimization (ISDPSO) algorithm maintains traditional velocity and position update formulas but incorporates two modifications. Firstly, it introduces a rounding operation in velocity updating to ensure integer values. Secondly, it incorporates a modulo operation in position updating to constrain values to integers within the range of the network's node count.

The update formulas of velocity and position in ISDPSO algorithm are as follows [114]:

$$v_{id} = round(\omega * v_{id} + c_1 * rand() * (pbest_i - x_{id}) + c_2 * rand() * (gbest - x_{id})) \tag{4.4}$$

$$x_{id} = (x_{id} + v_{id}) \bmod N + 1 \tag{4.5}$$

$$\omega = \frac{\omega_{max} - \omega_{min}}{TotalIter} \times CurIter \tag{4.6}$$

Where **N** represents the number of nodes in the network, $\mathbf{x_{id}}$ is an integer ranging from 1 to **N**, ω denotes the inertia weight, and ω_{max} and ω_{min} represent the maximum and minimum values

of ω respectively. **CurIter** denotes the current iteration count, while **TotalIter** represents the maximum number of iterations.

4.5.2 Discrete Particle Swarm Optimization with Redefined Operator

The position of an individual denotes a community detection configuration encompassing nodes and their neighbors, collectively forming a problem's search space. A fundamental challenge in enhancing discrete PSO algorithms lies in representing an individual's velocity and position, as will be elaborated on later. Another pivotal challenge involves redefining velocity and position update formulas to align with the discrete representation of the community detection problem within the basic PSO framework. To address this, they [114] introduce the Improved Discrete Particle Swarm Optimization with Redefined Operator (IDPSO-RO) algorithm.

The update equations for velocity and position in the IDPSO-RO algorithm are outlined below [114]:

$$\begin{aligned} v_{id} &= \omega \otimes v_{id} \oplus c1 \otimes (p_{id} \ominus x_{id}) \oplus c2 \otimes (p_{gd} \ominus x_{id}) \\ x_{id} &= x_{id} \oplus v_{id} \end{aligned} \tag{4.7}$$

The operational rules for the velocity and position update formulas in the IDPSO-RO algorithm are specified as follows [114]:

I. **Velocity Update (Definition 1 - Operator $\otimes$):** When the velocity V is multiplied by a constant c, it will generate a random number within the range of $[0, 1]$. If the random number is less than c, then the result is V, otherwise, the result is 0.

$$c \otimes V = \begin{cases} V, & \text{if rand}(0,1) < c \\ 0, & \text{otherwise} \end{cases}$$

Pseudo code:

Algorithm 1 Operator $\otimes$

Input: c, V
Output: Result
if *Random()* $< c$ **then**
    Result $\leftarrow$ V
end
Result $\leftarrow$ 0

Exemple:

If we have c = 0.5, V_1 ,where each value in the array represent a node velocity,And if

$rand() > c$ in the case node 2(value 3), the result would be :

V_1	8	6	3	7	**Result:**	8	6	0	7

II. **Velocity and Position Update (Definition 2 - Operator $\oplus$):** This operator can be applied to add velocities and velocity or add a position and velocity. The result of adding V_1 and V_2 is either V_1 or V_2:

$$V_1 \oplus V_2 = \begin{cases} V_1, & \text{if } (V_1 \neq 0 \text{ and } V_2 = 0) \text{ or} \\ & \quad (V_1 \neq 0 \text{ and } V_2 \neq 0 \text{ and rand}(0,1) < 0.5) \\ V_2, & \text{otherwise} \end{cases}$$

Pseudo code:

Algorithm 2 Operator $\oplus$

Input: V1, V2
Output: Result

```
if V1 ≠ 0 and V2 = 0 then
    Result ← V1
else
    if V1 ≠ 0 and V2 ≠ 0 and Random() < 0.5 then
        Result ← V1
    else
        Result ← V2
    end
end
```

Exemple:

If we have V1, V2,where each value in the array represent a node velocity.

V_1	8	6	3	7	**Result:**	2	4	3	7
V_2	2	4	0	3					

- We have V1=8,V2=2, V1 and V2 not equal 0 , it mean we look at rand(0,1), if rand() < 0.5 the result would be V1=8, otherwise the result is V2=2.in our case its 2 because rand() > 0.5 ,same in the case V1=6,V2=4.
- in the case V1=3,V2=0 the result is V1=3 because V2=0 and V1≠0.
- in the case V1=7,V2=3.the result is 7 because (V1 and V2 ≠0) and rand() < 0.5

When updating an individual's position, the new position is calculated by adding the previous position to the current velocity. If the current velocity is 0, the position remains unchanged. However, if the velocity is non-zero, the position and velocity of the individual are swapped to update the position.

$$X_1 \oplus V_1 = \begin{cases} X_1, & \text{if } V_2 = 0 \\ X_1 \leftrightarrow V_1, & \text{if } V_1 \neq 0 \end{cases}$$

Pseudo code:

Algorithm 3 Update position $\oplus$

Input: $X1$, $V1$
Output: $X1$, $V1$

```
prev_position ← []
for each node in X1 do
    if V1[node] = 0 then
        X1[node] ← X1[node]
    else
        prev_position[node] ← X1[node]
        X1[node] ← V1[node]
        V1[node] ← prev_position[node]
    end
end
return X1, V1
```

Example:

V1 where each value in the array represent a node velocity,X1 where each value in the array represent a node position.

X_1	8	6	3	7
V_1	2	4	0	3

new X1:	2	4	3	3
new V1:	8	6	0	7

- In the case X1=8, V1=2, and V1≠0 , the result would be swap the position and the velocity, the result are shown in the new X1, new V1.
- In the case X1=6, V1=4, and V1≠0 , the result would be swap the position and the velocity, the result are shown in the new X1, new V1.
- In the case X1=3, V1=0, because V1=0 the result stay unchanged.
- In the case X1=6, V1=4,the result would be swap the position and the velocity because V1≠0 , the result are shown in the new X1, new V1.

III. **Position Subtraction (Definition 3 - Operator $\ominus$):** This is the inverse operation of Operator $\oplus$, and it is suitable for the subtraction of two positions. When position X_1 is not equal to X_2, the result is X_1; otherwise, the result is 0.

$$X_1 \ominus X_2 = \begin{cases} X_1, & \text{if } X_1 \neq X_2 \\ 0, & \text{if } X_1 = X_2 \end{cases}$$

Pseudo code:

Algorithm 4 Operator subtract $\ominus$

Input: $X1, X2$
Output: $result$

```
result ← []
for each node in X1 do
    if X1[node] ≠ X2[node] then
        result[node] ← X1[node]
    else
        result[node] ← 0
    end
end
return result
```

Example:
If we have X1 , X2 where each value in the array represent a node position.

X_1	2	6	3	3
X_2	2	4	3	7

result:

0	6	0	3

- In the case X1=2,X2=2, we have X1 = X2 so by applying the previous definition the result would be 0.
- In the case X1=6,X2=4, we have X1 $\neq$ X2 so by applying the previous definition the result would be X1=6.
- In the case X1=3,X2=3, we have X1 = X2 so by applying the previous definition the result would be 0.
- In the case X1=3,X2=7, we have X1 $\neq$ X2 so by applying the previous definition the result would be X1=3.

Community detection using the PSO algorithm doesn't require prior knowledge of community size or quantity. Typically, community modularity (which we saw in Chapter 3) serves as the PSO algorithm's fitness function, with the individual yielding maximum modularity chosen as the optimal solution. Each individual in the population corresponds to nodes in the network, encoded as an N-dimensional variable X_i ($X_i = (x_{i1}, x_{i2}, \ldots, x_{id}, \ldots, x_{iN}); 1 \leq x_{id} \leq N$) representing a community detection scheme. Each dimension of X_i signifies a node, If the value of the i^{th} dimension is j, it indicates that node i and node j belong to the same community.

During population initialization, each node identifies its network neighbors based on the connections between nodes. The position value of an individual's i^{th} dimension corresponds to the serial number of a randomly selected node from its neighbors.

Tableau 4.1: A visual representation illustrating how individuals are encoded in PSO

Individual	**1**	**2**	**3**	**4**	**5**	**6**
Position	2	1	1	5	6	5

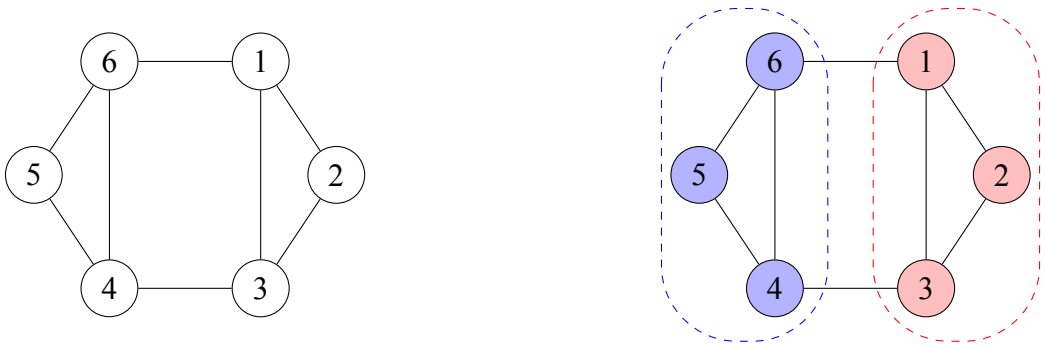


Figure 4.1: Schematic diagrams of the network and the community detection result

4.5.2.1 Fitness function and Pseudo code:

Community modularity (Q) is often used to compare different community detection results,which has been described in detail in chapter 3 (3.3.5), and it also can be regarded as the fitness function in optimization algorithms.

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i \cdot k_j}{2m} \right) \delta(c_i, c_j) \tag{4.8}$$

$$\delta(c_i, c_j) = \begin{cases} 1, & \text{if nodes } i \text{ and } j \text{ are in the same community,} \\ 0, & \text{otherwise.} \end{cases}$$

Algorithm 5 Calculate_fitness(modularity Q)

Input: particle_position
Output: Q
Initialize num_nodes to the number of nodes in the graph
Calculate the number of edges in the graph and store it in m
Create an array 'degrees' containing the degree of each node in the graph
Initialize Q to 0
for *each edge* $(node_i, node_j)$ *in the graph* **do**
    Calculate community_delta: 1 if particle_position[node_i] equals particle_position[node_j], else 0
    Update Q: $Q+ =$ (1 if the graph has an edge between node_i and node_j, else 0) $-$ $\frac{\text{degree of node\_i} \times \text{degree of node\_j}}{2 \times m} \times$ community_delta
end
Q: $Q/ = 2 \times m$
return Q

4.5.2.2 community correction strategy

The PSO algorithm for community detection can be sensitive to network structure due to the inherent randomness in updating velocities and positions. This can lead to unstable outcomes and fragmented communities. To improve accuracy, a **community correction strategy** leveraging network connectivity characteristics is proposed in reference [114].

In the PSO algorithm, each dimension of an individual's position corresponds to a node in the network. If the ith dimension of the position is j, it implies that node i and node j belong to the same community. The community correction strategy identifies the communities to which all nodes in the network are assigned based on the best individual's position obtained. Subsequently, it corrects each dimension of the best individual's position, ensuring correction for every node in the network. To prevent potential entrapment in local optima, this correction is applied after specific iterations: $1/4$ of the $TotalIter$, $1/2$ of the $TotalIter$, $3/4$ of the $TotalIter$, and at the $TotalIter$.

In the **community correction strategy**, a node's community assignment depends on **community influence**, determined by the proportion of community C among its neighbors. This influence reflects the theory of community detection, where the number of edges within a community exceeds those between nodes outside it. The community influence CommInf$(A; C)$ for node A is calculated based on the proportion of community C in its neighbors ,calculated as [114]:

$$N(A) = \sum_{l} n(A; C). \tag{4.9}$$

$$\text{CommInf}(A, C) = 1 - e^{-\frac{n(A,C)}{N(A)}}. \tag{4.10}$$

$n(A;C)$ represents the number of nodes that are neighbors of node A in community C. $N(A)$ is a fixed value for node A, so the community influence $CommInf(A;C)$ increases as $n(A;C)$ increases. $CommInf(A;C)$ ranges from 0 to 1.

The community correction strategy in the PSO algorithm iteratively examines all dimensions of the best individual's position. Each dimension corresponds to a node in the network. Community influence, calculated using equation 4.10, determines the community Maxl with the highest influence. Nodes within Maxl are randomly assigned serial numbers to the corrected position dimension, updating the best individual's position for the next iteration. This strategy leverages network connectivity knowledge, accelerating PSO convergence and reducing search time.

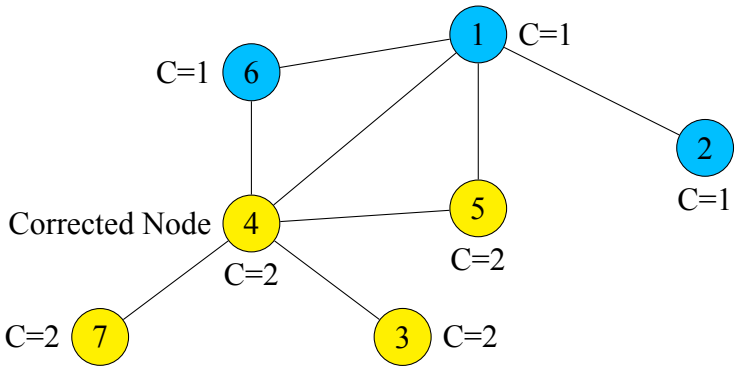


Figure 4.2: Schematic diagram of community correcting strategy

Algorithm 6 Community Influence Calculation

Input: Graph G, Node, List of $communities$
Output: $influence$ of each node in each community

```
Initialize
    influence ← List(int)
    neighbors ← list of neighbors
    total_neighbors ← length of neighbors
    foreach neighbor ∈ neighbors do
        foreach (community) ∈ (communities) do
            if neighbor ∈ community then
                influence[i] ← influence[i] + 1
            end
        end
    end
end
foreach (i, count) ∈ influence do
    influence[i] ← 1 − exp(−count/total_neighbors)
end
return influence
```

Bellow **Algorithm 6 [114]** outlines the process of a community detection method incorporating a community correction strategy.

Algorithm 7 A novel community detection method based on ISDPSO and IDPSO-RO algorithms

1: Initiate the number of nodes N in the network and configure the parameters for the PSO algorithm;
2: Create a population consisting of N individuals with random velocities and positions in D dimensions, based on the connections among the nodes in the network;
3: Set current number of iterations $CurIter$ to 1;
4: **while** The termination condition is not satisfied ($CurIter \neq TotalIter$) **do**
5: Calculate each individual's fitness value;
6: Compare the individual's fitness value with its p_{best_i}. If the current fitness is better than p_{best_i}, then assign p_{best_i} to the current fitness and assign the current position to p_i;
7: Find the individual with the best fitness value. If its fitness is better than $gbest$, then set p_g equal to the individual's position;
8: Update the velocity and position of each individual;
9: **if** $CurIter = \frac{1}{4}TotalIter$ or $\frac{1}{2}TotalIter$ or $\frac{3}{4}TotalIter$ or $TotalIter$ **then**
10: Find the community for each node based on the position of the best individual;
11: Calculate the community influence of each node;
12: Correct each dimension of the best individual based on community correcting strategy;
13: **end if**
14: Let $CurIter = CurIter + 1$;
15: **end while**

4.6 A modified IDPSO-RO

4.6.1 The architecture of the Proposed System

Our algorithm consists of three main parts:

1) Initialisation of parameters,population,position,velocity and real dataset.

2) Fitness Evaluation and Update velocity,position,personal best,global best,then apply local search strategy to global best.then we compare,the fitness of the corrected gbest by local search strategy and the fitness of the current gbest,we select the best bettwen them.

3) Apply majority voting to the final gbest at specific iterations to avoid local minimum.then find communities based on the corrected gbest by majority voting.

The overall architecture of our system is shown in the Figure. 4.3:

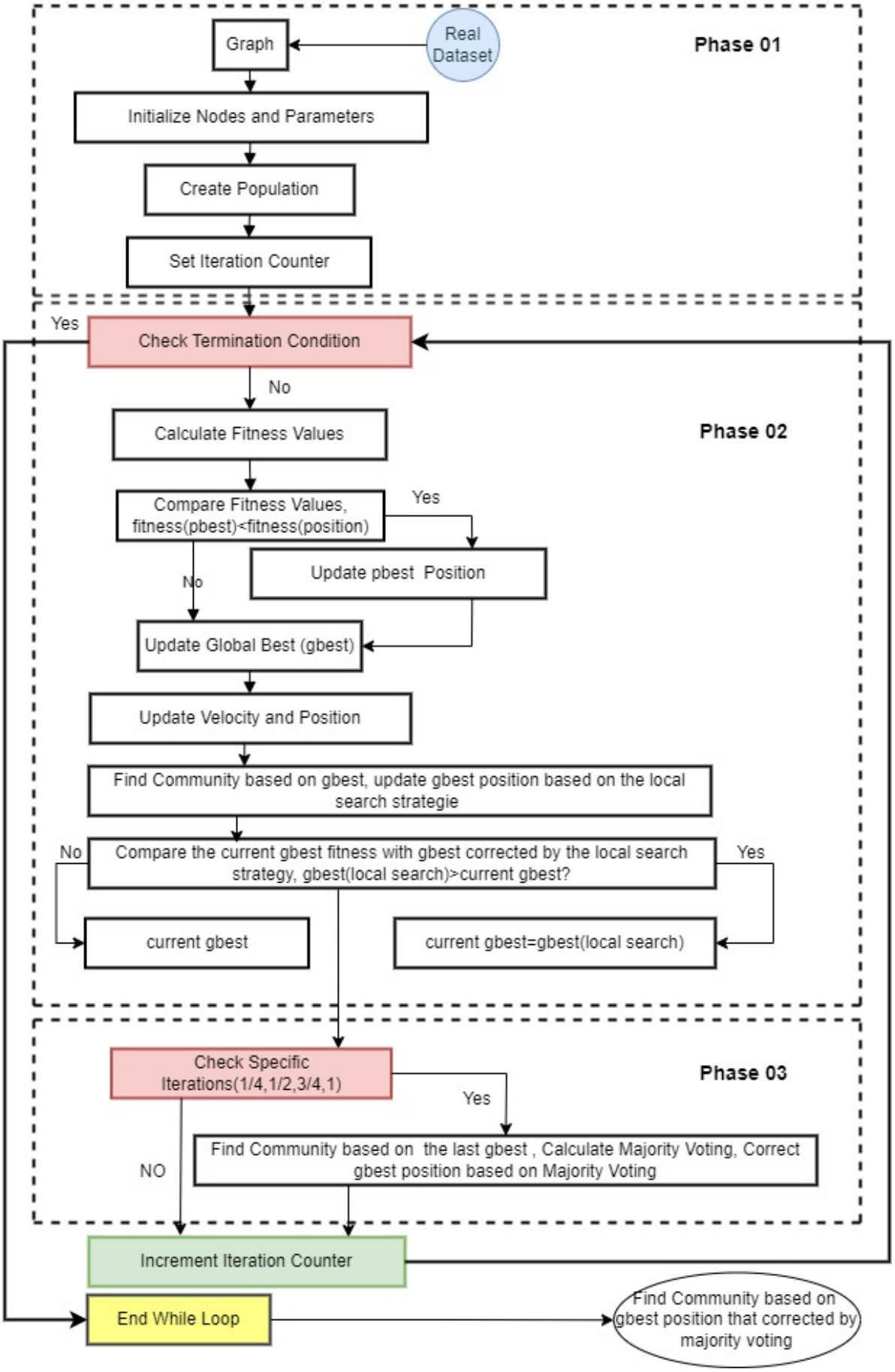


Figure 4.3: General architecture of the system

Our approach is based on the Modularity Density as fitness function,local search strategy, and Majority Voting (We'll explain them all below).

- **Phase 01: Initialization**
  This phase involves setting up the initial conditions for the algorithm.First we read the dataset and then we convert it into a graph. It includes initializing the network nodes and parameters, creating the initial population with random positions and velocities, and setting the iteration counter.

- **Phase 02: Fitness Evaluation and Update**
  In this phase, the algorithm evaluates the fitness of each individual, updates personal best positions if better solutions are found, and determines the global best position. The velocities and positions of individuals are then updated accordingly.And then find community based on gbest position. Based on the found communities we will correct gbest position by local search strategy,then we compare,if the fitness of the corrected gbest by local search strategy greater than fitness of the current gbest, then we choice the gbest of the local search strategy.otherwise, we choice the current gbest

- **Phase 03: Community Detection and Correction**
  This phase focuses on community detection and refinement. At specific iterations, the algorithm checks for community structure, calculates the majority voting of each node, and corrects the global best position based on majority voting. The iteration counter is incremented, and the process repeats until termination conditions are met.Finally finds the community based on the corrected gbest position by majority voting.

4.6.2 Modularity Density

According to [126], another form of modularity Q is as follows:

$$Q = \sum_{i=1}^{N} \left[\frac{L(V_i, V_i)}{L(V, V)} - \left(\frac{L(V_i, V)}{L(V, V)} \right)^2 \right] \tag{4.11}$$

where:

N is the number of communities,
$L(V_k, V_n)$ represents the sum of the edge weights between nodes in community V_k and nodes in community V_n, defined as:

$$L(V_k, V_n) = \sum_{i \in V_k, j \in V_n} A_{ij}$$

where A_{ij} is the adjacency matrix of the network.
$L(V, V)$ represents the total edge weight in the network, given by:

$$L(V, V) = 2m$$

where m is the total number of edges in the network.
$L(V_i, V)$ represents the sum of the edge weights between nodes in community V_i and all nodes in the network, defined as:

$$L(V_i, V) = \sum_{j \in V_i, k \in V} A_{jk}$$

A class of methods aimed at maximizing modularity has been developed. However, modularity has the disadvantage of resolution limits because it contains an intrinsic scale that depends on the size of the links in the network. When the modules are smaller than this scale, modularity cannot detect them accurately.

To address this issue, **Li et al** . proposed modularity density (D) in [126]. Modularity density evaluates the partition of a network based on the concept of average modularity degree and overcomes the resolution limit in community detection:

$$D = \sum_{i=1}^{N} \frac{L(V_i, V_i) - L\left(V_i, \overline{V_i}\right)}{|V_i|}$$

where $L(V_i, V_i)/|V_i|$ and $L(V_i, \overline{V_i})/|V_i|$ represent the average internal and external degrees of the i-th community, respectively. D tries to maximize the difference between the internal and external degrees, thereby improving community detection accuracy.

Modularity density D is related to the density of subgraphs and provides a solution to the issue where Q is sensitive to the network size and the interconnections between modules. Therefore, D can be used to determine if the networks are partitioned into the correct communities. According to the definition of modularity density D, a larger value of D indicates a more accurate partition. Then Li et al. improved D to a general version by setting a parameter λ to the proportion of average internal degree and external degree:

$$D_\lambda = \sum_{i=1}^{N} \frac{2\lambda L(V_i, V_i) - 2(1-\lambda) L(V_i, \overline{V_i})}{|V_i|} \tag{4.12}$$

D_λ is a convex combination of ratio cut and ratio association. It aims to maximize the density of links within a community while minimizing the density of links between different communities. When $\lambda = 1$, D_λ is equivalent to ratio association; when $\lambda = 0$, D_λ is equivalent to ratio cut; and when $\lambda = 0.5$, D_λ equals D. Using smaller values of λ allows the decomposition of the network into larger communities, whereas larger values of λ result in smaller communities. This approach enables the discovery of more detailed structures and multiple levels within the network. In this modified IDPSO-RO we use the Modularity Density D_λ as fitness function.

Algorithm 8 Calculate Fitness (modularity Density)

Input: particle_position
Output: D_lambda

```
Function calculate_fitness(particle_position):
    communities ← []
    L_Vi_Vi ← []  L_Vi_Vibar ← []
    for community in communities do
        internal_degree_sum ← 0 ; external_degree_sum ← 0 ;
        for node in community do
            for neighbor in G.neighbors(node) do
                if neighbor in community then
                    internal_degree_sum ← internal_degree_sum + 1
                end
                else
                    external_degree_sum ← external_degree_sum + 1
                end
            end
        end
        L_Vi_Vi.append(internal_degree_sum/2) ; L_Vi_Vibar.append(external_degree_sum)
    end
    D_lambda ← 0
    for internal, external in (L_Vi_Vi, L_Vi_Vibar) do
        community_size ← len(communities) ;
        if community_size > 0 then
            term1 ← (2 × lambda_param × internal)/community_size
            term2 ← (2 × (1 − lambda_param) × external)/community_size
            D_lambda ← D_lambda + (term1 − term2)
        end
    end
    return D_lambda
```

4.6.3 local search strategy

We plan to employ a local search strategy closer to the simulated annealing algorithm. Local search offers the advantage of efficiently discovering optimal solutions and speeding up convergence.

For a given solution $C = \{C_1, C_2, ..., C_l\}$, for each vertex v in C, we compute the connectivity of vertex v with community C_i (as shown in equation 4.13), where v belongs to C_i. Next, for the vertex v_{min} corresponding to the minimum connectivity, we compute its connectivity with all other communities C_j, where C_j belongs to C and $i \neq j$. Finally, we adjust vertex v_{min} to C_{max}, where v_{min} and C_{max} exhibit the maximum connectivity.

For each vertex y_j, $0 < j < n$ [127]:

$$Connect_j = f(v_i, C_j) \quad C_j \in C, 0 < j < k \tag{4.13}$$

where $C_1, C_2, \ldots, C_k$ is a community detection result and k is the number of communities. The function $f()$ calculates the edges between vertex v_i and communities C_j, where $0 < j < k$.If community max $C_{max}(C_{max} \in C)$ corresponds to the max Connect, adjust v_i to C_{max}.

Pseudo code:

Algorithm 9 Calculate Connect

Input: node, community
Output: connect
Function `calculate_connect(`*node, community*`):`
    return $\sum_{neighbor \in G.neighbors(node)}(1$ *if neighbor in community else* $0)$

Algorithm 10 Local Search strategy

Input: gbest_position
Output: gbest_position
Function `local_search(`*position*`):`
    communities ← community(position)
    for *community* ***in*** *communities* **do**
        v_min ← min(community, calculate_connect(node, community))
        max_connect ← None
        best_community ← None
        for *other_community* ***in*** *communities* **do**
            if *other_community* $\neq$ *community* **then**
                connect_value ← calculate_connect(v_min, other_community)
                if *connect_value* $>$ *max_connect* **then**
                    max_connect ← connect_value
                    best_community ← other_community
                end
            end
        end
        if *best_community* $\neq$ *None* **then**
            community.remove(v_min)
            max_connect_community=best_community.add(v_min)
            gbest_position[v_min]=max_connect_community
        end
    end
    return *gbest_position*

We gonna use this strategy to correct the gbest position.The steps to do that is bellow :

- **Getting Communities:**
  The function first obtains the communities based on the given position. This could be a method community(As in our case) that extracts communities from the current gbest position.

- **Finding Node with Minimum Connection:**
  It then iterates over each community in the list of communities.For each community, it

finds the node (v_{min}) that has the minimum connection to other nodes within the same community. This is done by calculates the connection for each node.

- **Determining Best Community to Move Node:**
  After finding v_{min}, it initializes variables max_connect and best_community. Then, it iterates over all other communities (other_community) except the current one (community).

- **Finding Best Community:**
  For each other_community, it calculates the connection value between v_{min} and nodes in other_community. If this connection value is greater than the maximum connection encountered so far (max_connect), it updates max_connect and best_community.

- **Moving Node to Best Community:**
  If a best_community is found (i.e., not None), it removes v_{min} from its current community and adds it to the best_community.

- **Returning Updated Position:**
  Finally, it returns the updated gbest position after the local search operation.

4.6.4 Majority Voting

Given a graph $G = (V, E)$ where V is the set of nodes and E is the set of edges, and a set of communities $\{C_1, C_2, \ldots, C_k\}$ the majority voting method assigns each node to the community label that is most frequent among its neighbors.
where:

- $v \in V$ is a node in the graph.
- $N(v)$ is the set of neighbors of node v.
- C_i is a community, and $c(v)$ denotes the community label of node v.

Steps:

1. **Count Community Labels Among Neighbors:**
   For each node $v \in V$, count the occurrences of each community label among its neighbors:

$$\text{count}_i(v) = \sum_{u \in N(v)} \delta(c(u), C_i) \tag{4.14}$$

   where $\delta(c(u), C_i)$ is the indicator function:

$$\delta(c(u), C_i) = \begin{cases} 1 & \text{if } c(u) = C_i \\ 0 & \text{otherwise} \end{cases}$$

2. **Assign the Most Frequent Community Label:**
   Assign node v to the community label that is most frequent among its neighbors:

$$c(v) = \arg\max_i \text{count}_i(v)$$

   If there is a tie (multiple community labels have the same highest count), a random choice among the tied labels can be made.

3. **Example:**
   Consider a simple example where node v has neighbors u_1, u_2, u_3, and their community labels are $c(u_1) = C_1$, $c(u_2) = C_1$, and $c(u_3) = C_2$. The counts would be:

$$\text{count}_1(v) = \delta(c(u_1), C_1) + \delta(c(u_2), C_1) + \delta(c(u_3), C_1) = 1 + 1 + 0 = 2$$

$$\text{count}_2(v) = \delta(c(u_1), C_2) + \delta(c(u_2), C_2) + \delta(c(u_3), C_2) = 0 + 0 + 1 = 1$$

   Since $\text{count}_1(v) > \text{count}_2(v)$, node v will be assigned to community C_1:

$$c(v) = C_1$$

Pseudo code:

Algorithm 11 Majority Voting

Input: node, communities
Output: community

```
Function majority_voting(node, communities):
    neighbor_communities ← []
     for neighbor in G.neighbors(node) do
        for idx, community in (communities) do
            if neighbor in community then
                neighbor_communities[idx] += 1
            end
        end
    end
    if neighbor_communities then
        return community with maximum count in neighbor_communities
    end
    else
        return random choice from list of community indices
    end
```

By using the Majority Voting, each node v will be assigned to community C_k since it has the most neighbors in that community. This approach helps in clustering nodes into communities based on local neighborhood information, promoting coherent community structure.

4.6.5 Algorithm of modified IDPSO-RO

Algorithm 12 A modified community detection method based on IDPSO-RO algo

1: Initiate the number of nodes N in the network and configure the parameters for the PSO;
2: Create a population consisting of N individuals with random velocities and positions in D dimensions, based on the connections among the nodes in the network;
3: Set current number of iterations $CurIter$ to 1;
4: **while** The termination condition is not satisfied ($CurIter \neq TotalIter$) **do**
5: Calculate each individual's fitness value;
6: Compare the individual's fitness value with its p_{best_i}. If the current fitness is better than p_{best_i}, then assign p_{best_i} to the current fitness and assign the current position to p_i;
7: Find the individual with the best fitness value. If its fitness is better than $gbest$, then set p_g equal to the individual's position;
8: Update the velocity and position of each individual;
9: Update $gbest$ based on the local search strategy
10: Compare the current $gbest$ fitness with $gbest$ corrected by the local search strategy,If its fitness is better than the current $gbest$,then set $gbest$ equal to the $gbest$ that corrected by local search strategy
11: **if** $CurIter = \frac{1}{4}TotalIter$ or $\frac{1}{2}TotalIter$ or $\frac{3}{4}TotalIter$ or $TotalIter$ **then**
12: Find the community for each node based on the position of the best individual that corrected by local search strategy;
13: Calculate the Majority Voting of each node in each community;
14: Correct each dimension of the best individual based on Majority Voting strategy;
15: **end if**
16: Let $CurIter = CurIter + 1$;
17: **end while**

Chapter 5

Implimentation and Experimental results

5.1 Introduction

In this chapter, we introduce the tools and languages utilized to implement the method proposed in the previous chapter. Following this, we present the results of our method in comparison with some of the most popular algorithms.

5.2 Working environment

5.2.1 Hardware environment

All our installations and tests are carried out on a computer with the following characteristics:

- **Processeur :** Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz.
- **Installed RAM:** 8.00 GB.
- **System type :**Windows 11 Pro Education,64-bit operating system, x64-based processor

5.2.2 Software environment

We utilized Python, version 3.11.1, for our implementation. Python is an interpreted, interactive, object-oriented, high-level programming language created by Guido van Rossum and first released in 1991 [128].

- **Python is Interpreted:** meaning it is processed at runtime by the interpreter. This eliminates the need to compile the program before execution, similar to languages like PERL and PHP.
- **Python is Interactive:**allows you to engage directly with the interpreter at its prompt, enabling you to craft programs in real-time.

- **Python is Object-Oriented:**Python embraces the Object-Oriented paradigm, facilitating the encapsulation of code within objects for streamlined programming.

Installing Python is free and easy, just download it from the website : https://www.python.org/downloads/ (See Figure. 5.1).

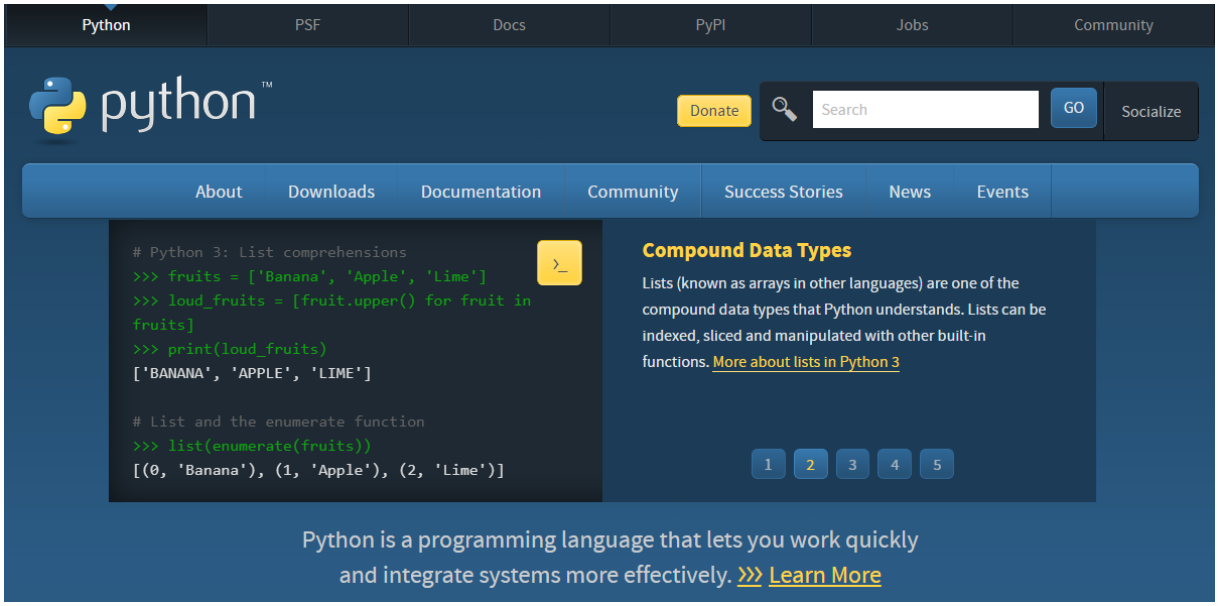


Figure 5.1: Python installation website

5.2.3 Platform & IDE

We used Visual Studio Code, commonly abbreviated as VS Code, is a source-code editor crafted by Microsoft for Windows, Linux, macOS, and web browsers. Its array of features encompasses debugging support, syntax highlighting, intelligent code completion, snippets, code refactoring, and integrated version control through Git. Users can customize themes, keyboard shortcuts, preferences, and augment functionality by installing extensions.

In the 2023 Stack Overflow Developer Survey (survey.stackoverflow.co/2023/most-popular-technologies-new-collab-tools) Visual Studio Code emerged as the top choice among 86,544 respondents for developer environment tools, with 73.71% indicating its usage.

To download and install the appropriate version of the VS code platform based on the user's computer operating system and the latest version of Python from the VS code website https://code.visualstudio.com/

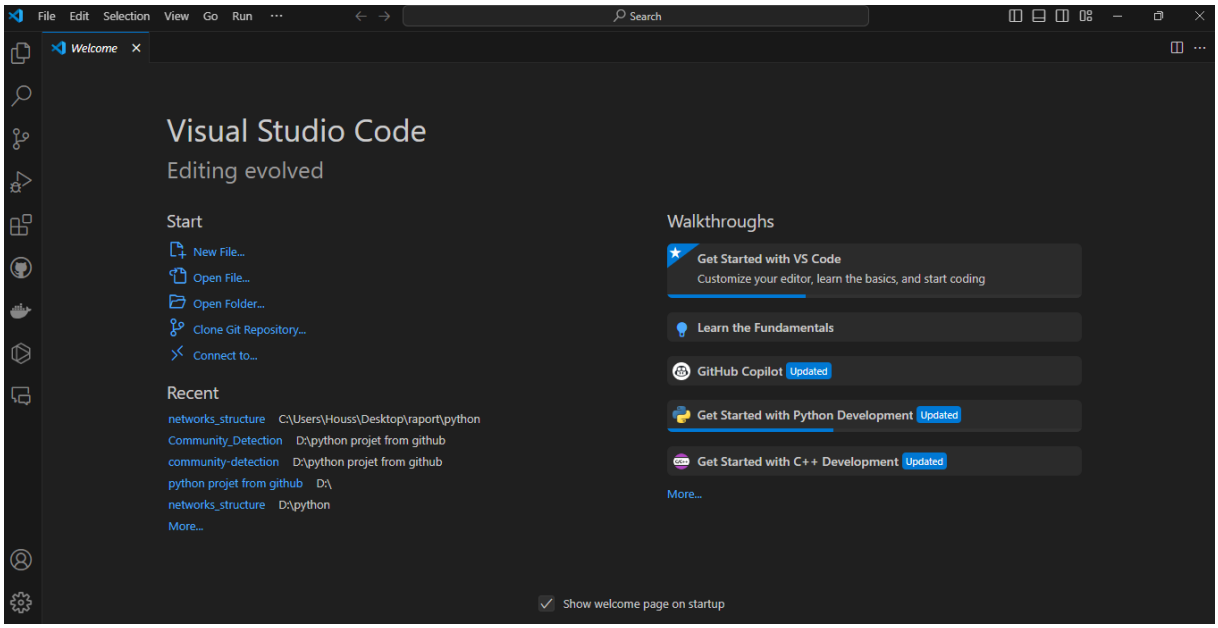


Figure 5.2: VS code

After the installation of VS code , We need install extensions to add new languages(in our case python), themes, debuggers, and to connect to additional services. Extensions run in separate processes, ensuring they won't slow down the editor.bellow our extensions installed:

- **Python extension for Visual Studio Code:**python language support with extension access points for IntelliSense (Pylance), Debugging (Python Debugger), linting, formatting, refactoring, unit tests, and more.
- **Pylance:** is an extension that works alongside Python in Visual Studio Code to provide performant language support. Under the hood, Pylance is powered by Pyright, Microsoft's static type checking tool. Using Pyright, Pylance has the ability to supercharge your Python IntelliSense experience with rich type information, helping you write better code faster.
- **Python Debugger:**A Visual Studio Code extension that supports Python debugging with debugpy. Python Debugger provides a seamless debugging experience by allowing you to set breakpoints, step through code, inspect variables, and perform other essential debugging tasks. The debugpy extension offers debugging support for various types of Python applications including scripts, web applications, remote processes, and multi-threaded processes.

5.2.4 Libraries using

A library is a set of predefined functions. These are grouped together and made available so that they can be used without having to be rewritten. Python is a very rich programming language

with its libraries.

We have used several packages (libraries) in this work, including :

Tableau 5.1: The libraries used in our work

Library	**Description**
NetworkX	A Python library designed for exploring graphs and networks, offering classes to represent graphical entities, generators to produce common graph structures, algorithms for network analysis, and foundational drawing utilities[networkx].
community	A Python library used for computing and measuring community structure[community].
NumPy	is a fundamental Python library for scientific computing, specializing in the manipulation of arrays, essentially vectors and matrices[numpy].
Matplotlib	this is a complete library of the Python programming language for plotting and visualizing data in graphical form (it can be used to create static, animated and interactive visualizations in Python)[matplotlib].
Random	This is a Python module that implements pseudo-random number generators for various distributions(Random variable generators)[random].
collections	This module implements specialized container datatypes providing alternatives to Python's general purpose built-in containers, dict, list, set, and tuple[collections].
sklearn	module includes functions to configure global settings and get information about the working environment[sklearn].
metrics	module includes score functions, performance metrics and pairwise metrics and distance computations[metrics].
sklearn.metrics.cluster	sub-module contains evaluation metrics for cluster analysis results. There are two forms of evaluation: supervised, unsupervised, which does not and measures the quality of the model itself[cluster].

To install the necessary packages, simply type the following codes into VS code Terminal :

- **NetworkX:** !pip install Networkx.
- **NumPy :** !pip install NumPy.

- **Matplotlib :** !pip install Matplotlib.

Downloading packages requires Internet access.

5.3 Experimental results and analysis

For a community detection algorithm to be considered efficient, the communities found must be relevant. In our work, the performance of the proposed algorithm has been assessed by an evaluation metric modularity Q (see 3.3.5), using four real-world networks 3.4 (karaty, dolphin, political books, American Football) , and comparing our results from the modified IDPSO_RO (MIDPSO-RO) with those of the algorithms described in section (3.3.4), namely Newman [94], Label Propagation [104] and the Louvain agorithm [129], and the original IDPSO-RO [114].

Tableau 5.2: Real network data sets

Network data sets	Nodes	Edges	Communities
Karate Club Network (Karate)	34	78	2
Dolphin Social Network (Dolphins)	62	159	2
American Politics Books Network (Polbook)	105	441	3
American football network (Football)	115	613	12

Tableau 5.3: Parameters of the algorithm

MIDPSO-RO		MIDPSO-RO	
Population size	**Iteration**	c_1, c_2, λ	ω
20	200	0.7	0.5

5.3.1 Karate Club Network results

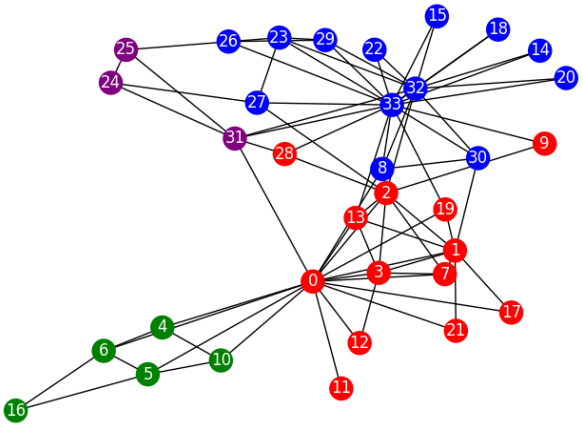


Figure 5.3: Community detection results on Karate data set using MIDPSO-RO algorithm execution 1

Figure. 5.3 show the community detection results in Karate network. Karate network actually has two communities,and **MIDPSO-RO** algorithm divides it into four communities with modularity Q=0.403 and its the maximum Q.

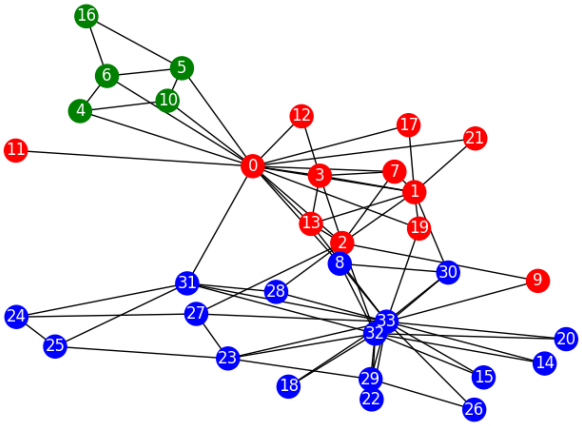


Figure 5.4: Community detection results on Karate data set using MIDPSO-RO algorithm execution 2

Figure. 5.4 show the community detection results in Karate network. Karate network actually has two communities,and **MIDPSO-RO** algorithm divides it into three communities with modularity Q=0.402.

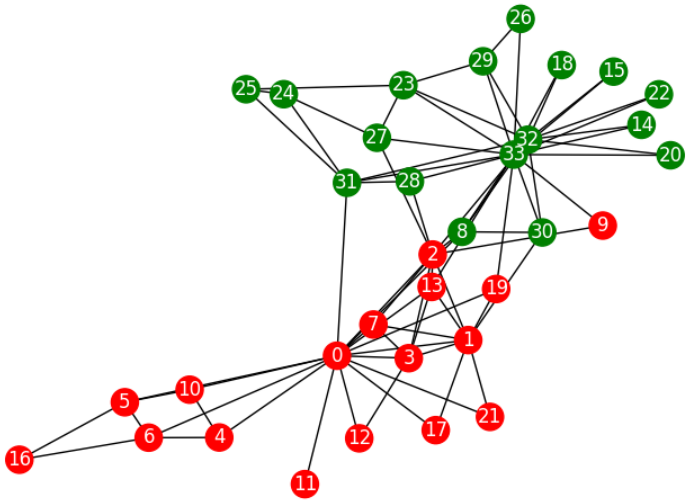


Figure 5.5: Community detection results on Karate data set using MIDPSO-RO algorithm execution 3

Figure. 5.5 show the community detection results in Karate network, MIDPSO-RO algorithm divides it into two communities with modularity Q=0.371.

5.3.2 Dolphin network results

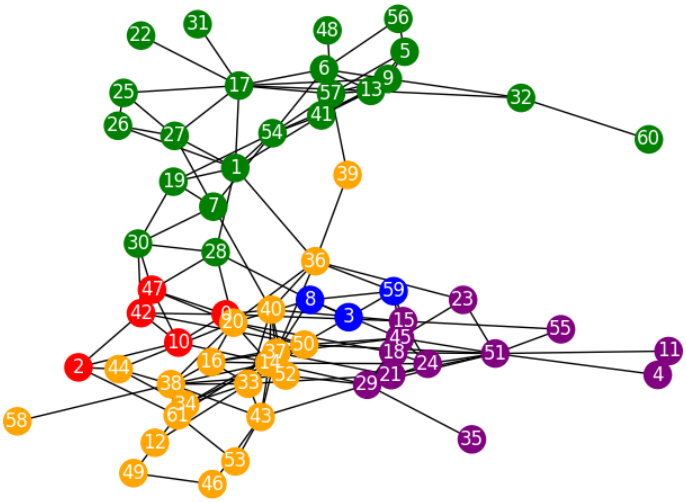


Figure 5.6: Community detection results on Dolphin data set using MIDPSO-RO algorithm execution 1

In Figure. 5.6, Dolphins network is detected by MIDPSO-RO algorithm with the maximum modularity Q=0.524, There are five communities in the result, while the actual network has two communities.

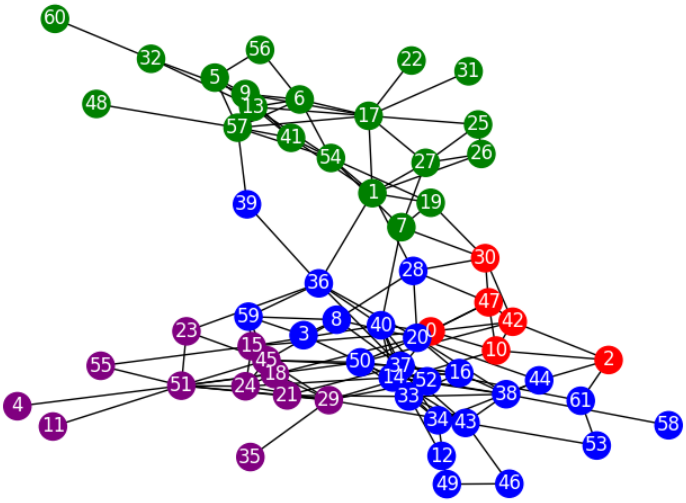


Figure 5.7: Community detection results on Dolphin data set using MIDPSO-RO algorithm execution 2

In Figure. 5.7, Dolphins network is detected by MIDPSO-RO algorithm with Q=0.509, There are four communities in the result.

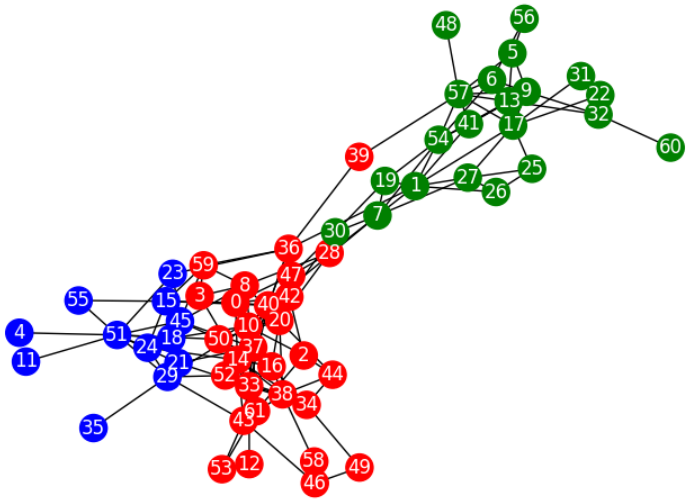


Figure 5.8: Community detection results on Dolphin data set using MIDPSO-RO algorithm execution 3

In Figure. 5.8, Dolphins network is detected by MSDPSO-RO algorithm with Q=0.478, There are three communities in the result.

5.3.3 American Politics Books Network

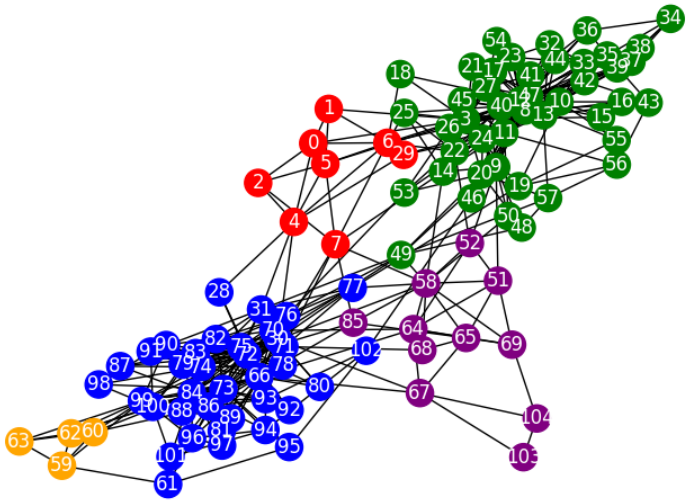


Figure 5.9: Community detection results on Polbook data set using MIDPSO-RO algorithm execution 1

In Polbook network, MIDPSO-RO algorithm have good performance. (Figure. 5.9) show good modularity obtained by MIDPSO-RO algorithm is Q=0.524 with 5 community while the actual network has three communities.

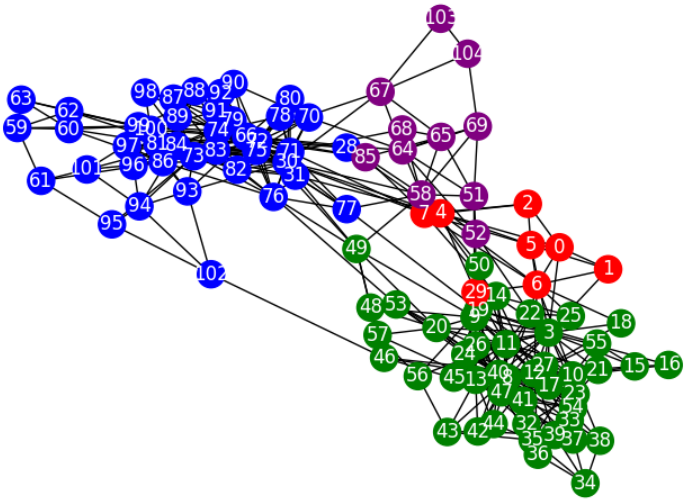


Figure 5.10: Community detection results on Polbook data using MIDPSO-RO algorithm execution 2

In Polbook network, MIDPSO-RO algorithm have good performance. (Figure. 5.10) show the best modularity obtained by MIDPSO-RO algorithm is Q=0.526 with 4 community while the actual network has three communities.

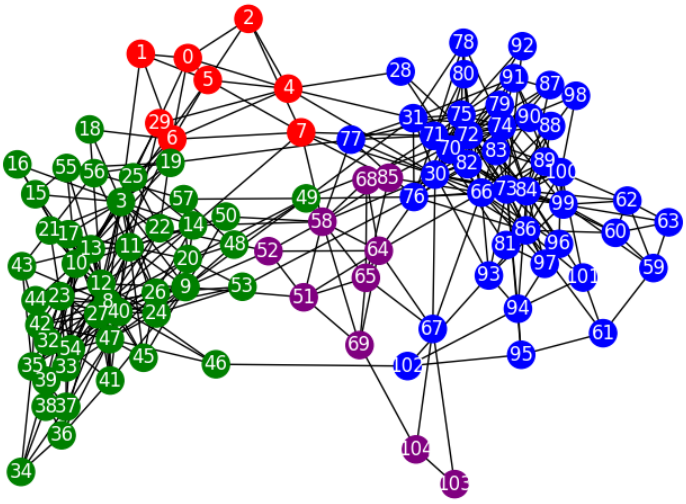


Figure 5.11: Community detection results on Polbook data using MIDPSO-RO algorithm execution 3

In Figure. 5.11, Polbook network is detected by MIDPSO-RO algorithm with Q=0.516, There are four communities in the result.

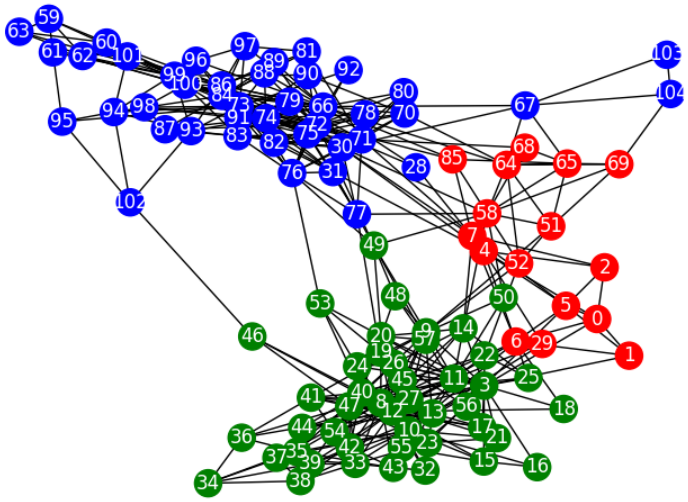


Figure 5.12: Community detection results on Polbook data using MIDPSO-RO algorithm execution 4

In Figure. 5.12, Polbook network is detected by MIDPSO-RO algorithm with Q=0.512, There are three communities in the result.

5.3.4 American football network

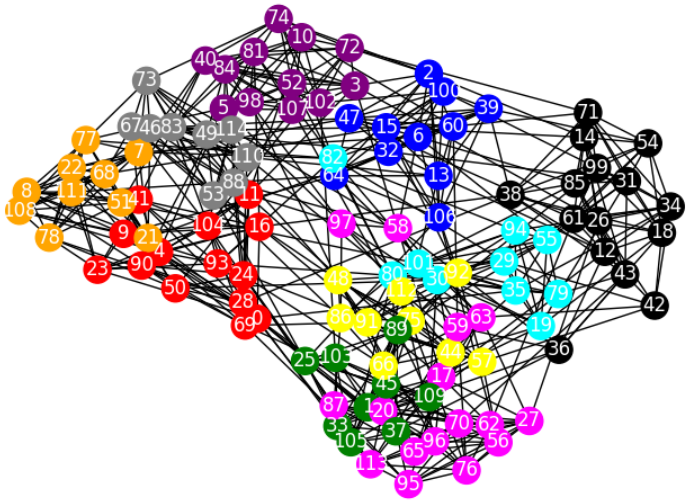


Figure 5.13: Community detection results on football dataset using MIDPSO-RO algorithm execution 1

In Figure. 5.13, football network is detected by MIDPSO-RO algorithm with the maximum Q=0.604, There are 10 communities in the result,while the actual network has 12 communities.

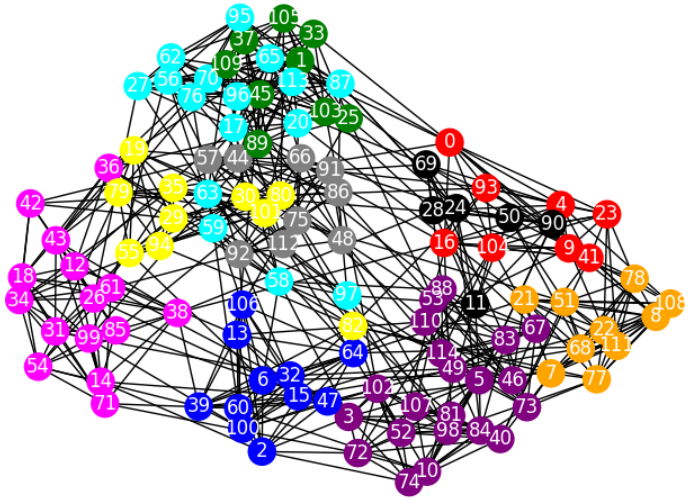


Figure 5.14: Community detection results on football dataset using MIDPSO-RO algorithm execution 2

In Figure. 5.14, football network is detected by MIDPSO-RO algorithm with Q=0.603, There are 11 communities in the result,while the actual network has 12 communities.

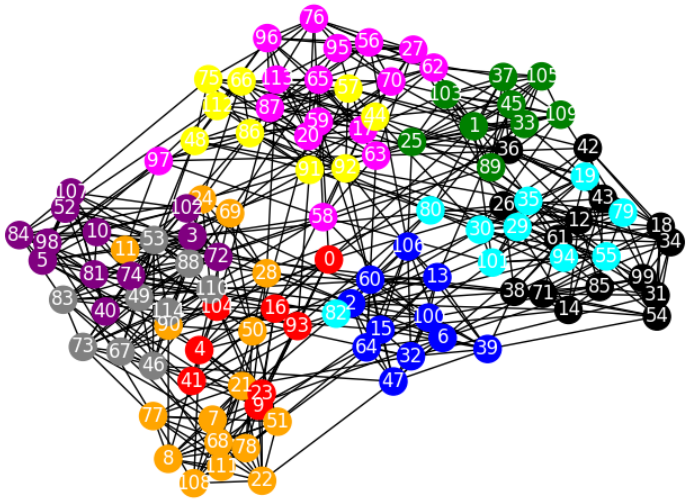


Figure 5.15: Community detection results on football dataset using MIDPSO-RO algorithm execution 3

In Figure. 5.15, football network is detected by MIDPSO-RO algorithm with Q=0.602, There are 10 communities in the result.

5.3.5 Comparison

The best modularity and NMI results obtained by IDPSO-RO and MIDPSO-RO algorithms except in football dataset which LPA (best NMI) are shown in Table 5.4, 5.6. As is shown in Figure. 5.16 and Figure. 5.17, our algorithm perform well. On the four data sets, the maximum modularity found by MISDPSO algorithm is better than GN and LPA algorithms. MIDPSO-RO

algorithm can get the maximum modularity among the LA algorithm in the Dolphins network, and the same modularity value in polbooks and football networks, LA better in Karate network. MIDPSO-RO algorithm can get the maximum modularity among IDPSO-RO in two networks Dolphins and Football and the same in Polbook network, IDPSO-RO better in Karate network.In the comparison of the value of NMI, MIDPSO-RO algorithm using Local search and Majority Voting perform best on dolphin and football data sets. On karate dataset, MIDPSO-RO algorithm can get the highest NMI amon GN LPA, and the next highest after LA. Therefore, Local search and Majority Voting strategy can effectively improve the accuracy of the detection. MIDPSO-RO algorithm can accurately get realistic results.

Tableau 5.4: Maximum modularity

Network	GN	LPA	LA	IDPSO-RO	MIDPSO-RO
Karate	0.359	0.112	0.418	**0.420**	0.403
Dolphins	0.378	0.456	0.518	0.520	**0.524**
Polbook	0.442	0.481	**0.526**	**0.526**	**0.526**
Football	0.400	0.593	**0.604**	0.602	**0.604**

Tableau 5.5: Corresponding values of normalized mutual information to maximum modularity

Network	GN	LPA	LA	IDPSO-RO	MIDPSO-RO
Karate	0.332	0.260	0.537	0.687	0.509
Dolphins	0.252	0.480	0.528	0.484	0.505
Polbook	0.247	0.543	0.459	0.441	0.388
Football	0.234	0.602	0.627	0.634	0.627

Tableau 5.6: Maximum normalized mutual information

Network	GN	LPA	LA	IDPSO-RO	MIDPSO-RO
Karate	0.332	0.260	0.550	**0.687**	0.547
Dolphins	0.252	0.480	0.530	0.546	**0.561**
Polbook	0.247	**0.543**	0.459	0.459	0.435
Football	0.234	0.602	0.627	0.634	**0.651**

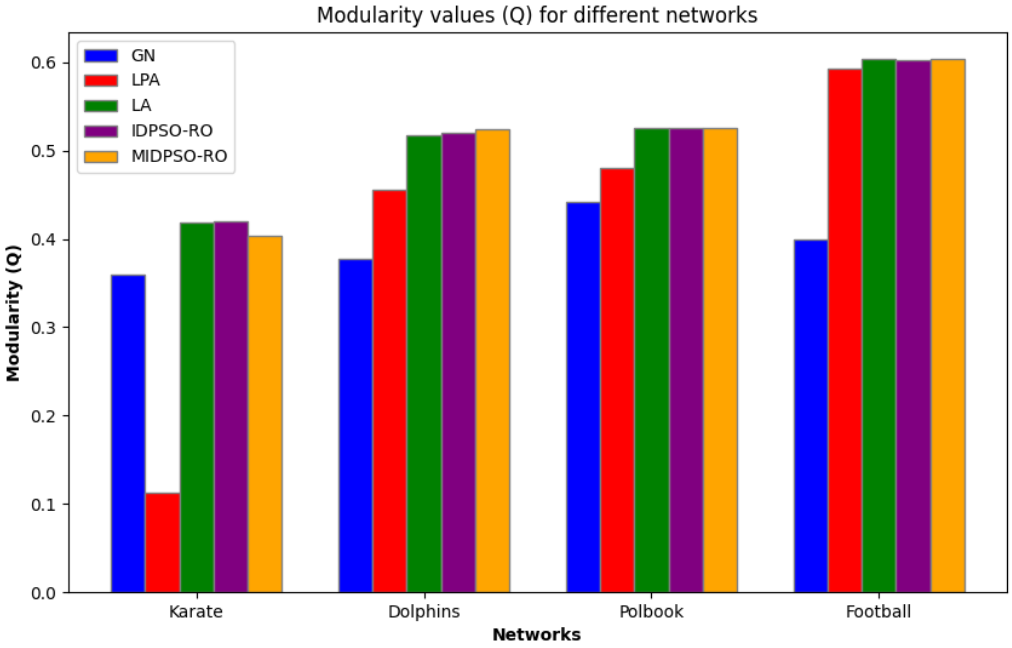


Figure 5.16: Comparisions of modularity

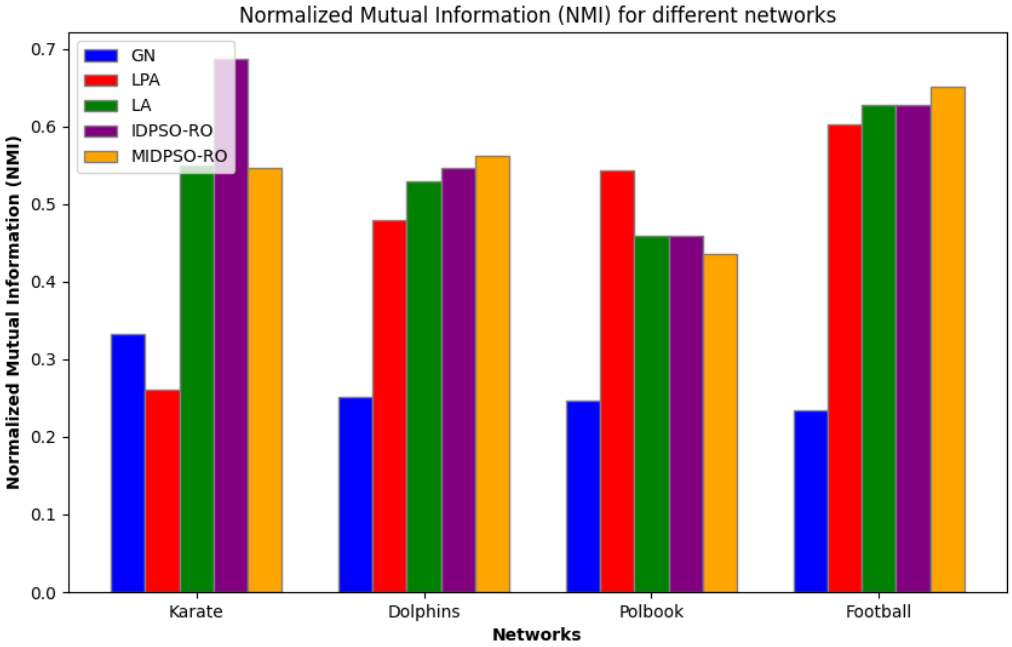


Figure 5.17: Comparisions of normalized mutual information

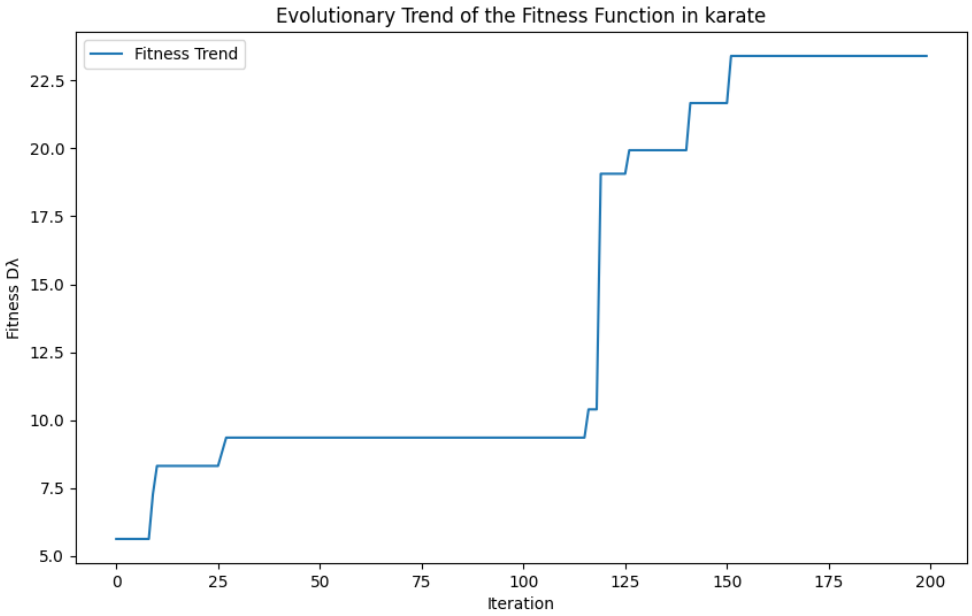


Figure 5.18: Evolutionary trend of MIDPSO-RO algorithm on Karate network

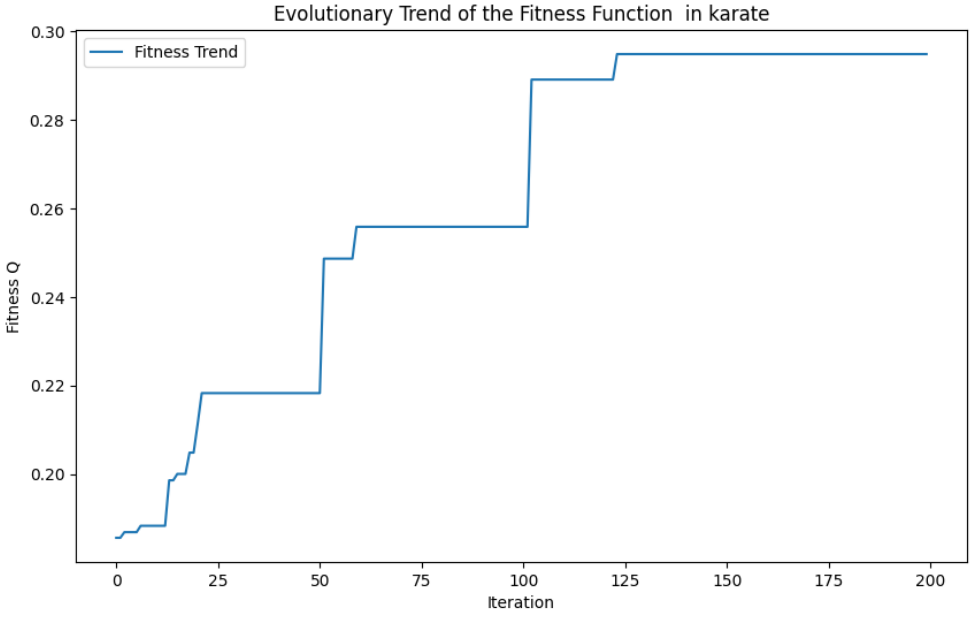


Figure 5.19: Evolutionary trend of IDPSO-RO algorithm on Karate network

Figures (5.18 ,5.19) depicts the convergence of IDPSO-RO and MIDPSO-RO algorithms on Karate network.As can be seen from the evolutionary trend, IDPSO-RO algorithm converges faster because Fitness Q reaches a stable value earlier (around 125 iterations) compared to fitness D_λ (around 150 iterations). MIDPSO-RO algorithm converges relatively slower in karate network.

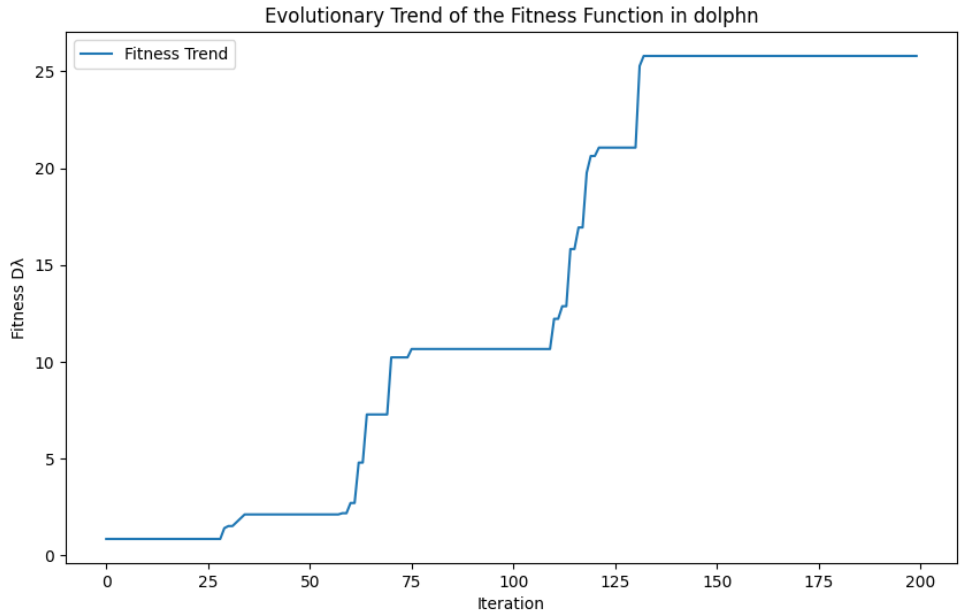


Figure 5.20: Evolutionary trend of MIDPSO-RO algorithm on dolphin network

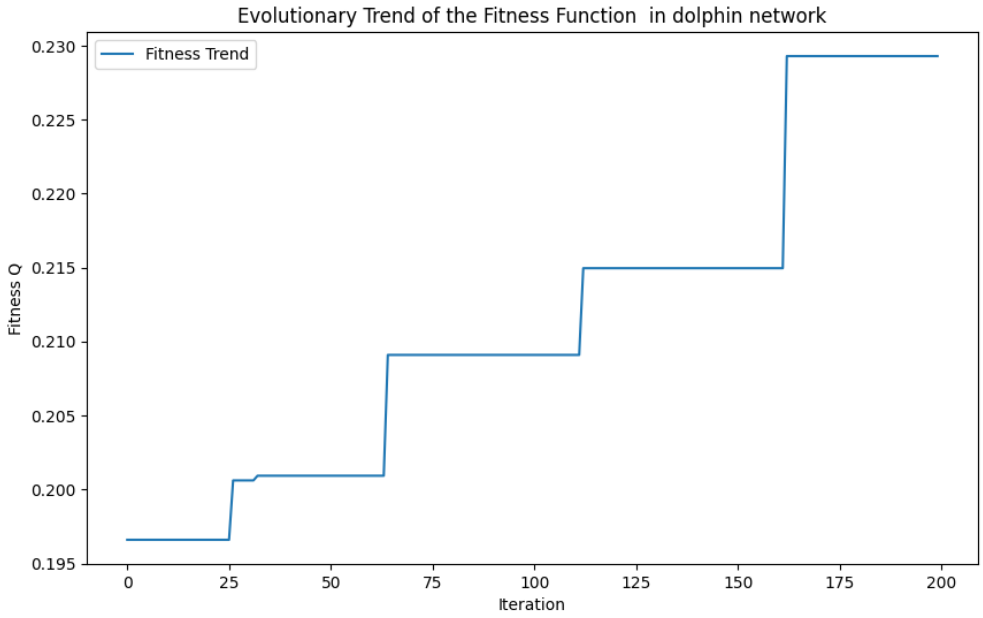


Figure 5.21: Evolutionary trend of IDPSO-RO algorithm on dolphin network

Figures (5.20, 5.21) depicts the convergence of IDPSO-RO and MIDPSO-RO algorithms on dolphin network. As can be seen from the evolutionary trend, MIDPSO-RO algorithm converges faster because Fitness D_λ reaches a stable value earlier (around 125 iterations) compared to fitness Q (around 175 iterations).

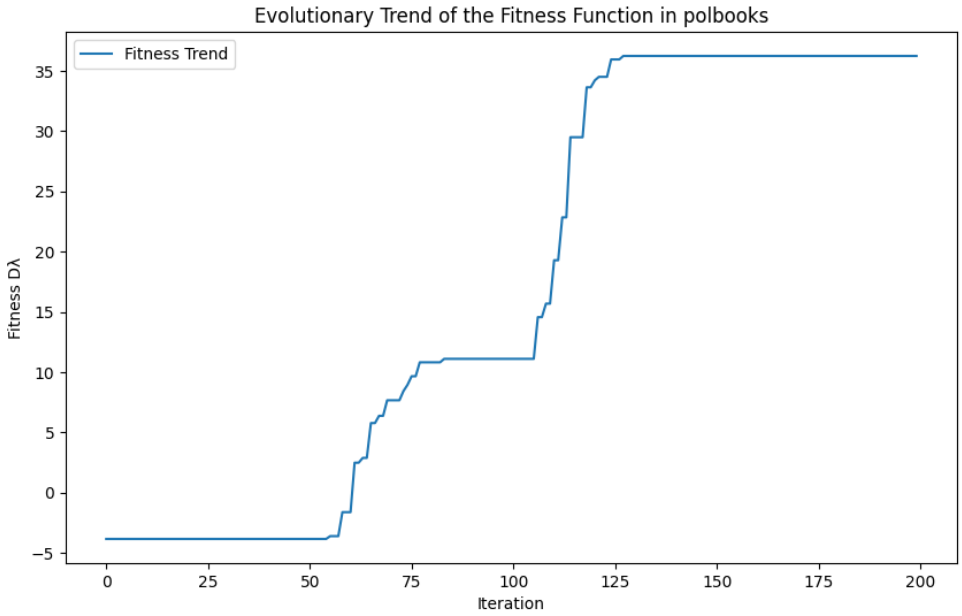


Figure 5.22: Evolutionary trend of MIDPSO-RO algorithm on polbooks network

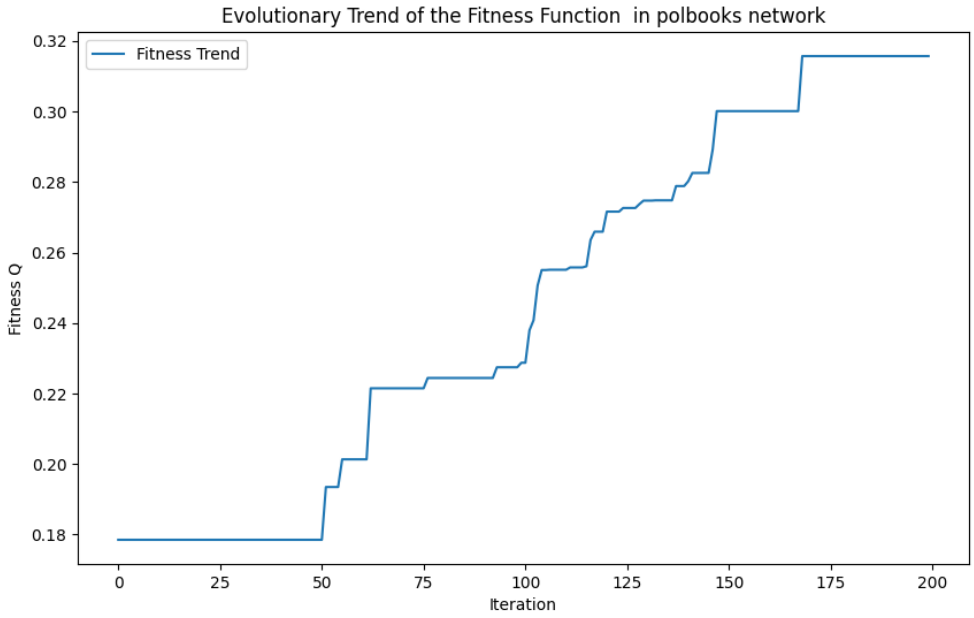


Figure 5.23: Evolutionary trend of IDPSO-RO algorithm on polbooks network

Figures (5.22, 5.23) depicts the convergence of IDPSO-RO and MIDPSO-RO algorithms on polbooks network. As can be seen from the evolutionary trend, MIDPSO-RO algorithm converges faster because Fitness D_λ reaches a stable value earlier (around 125 iterations) compared to fitness Q (around 175 iterations).

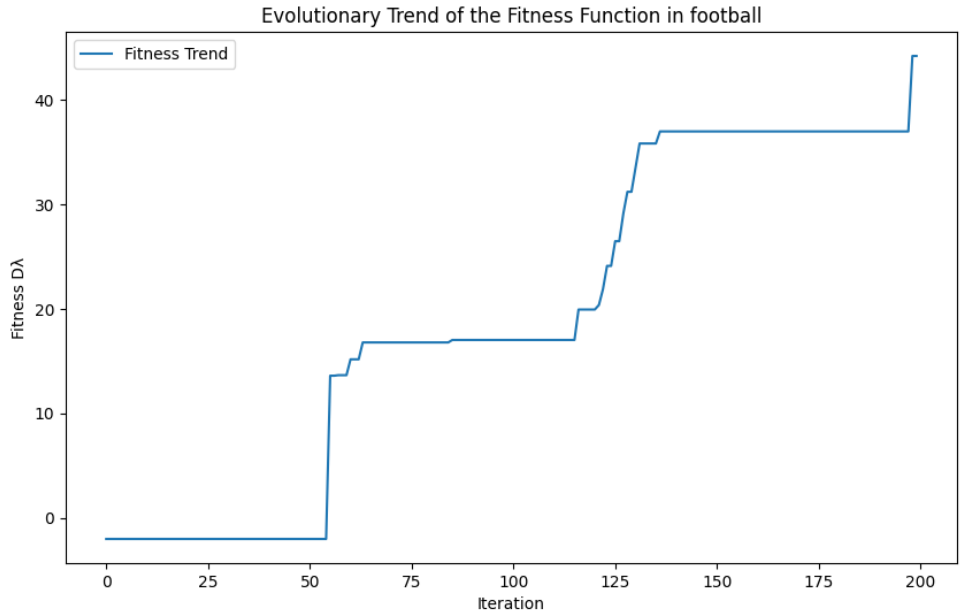


Figure 5.24: Evolutionary trend of MIDPSO-RO algorithm on football network

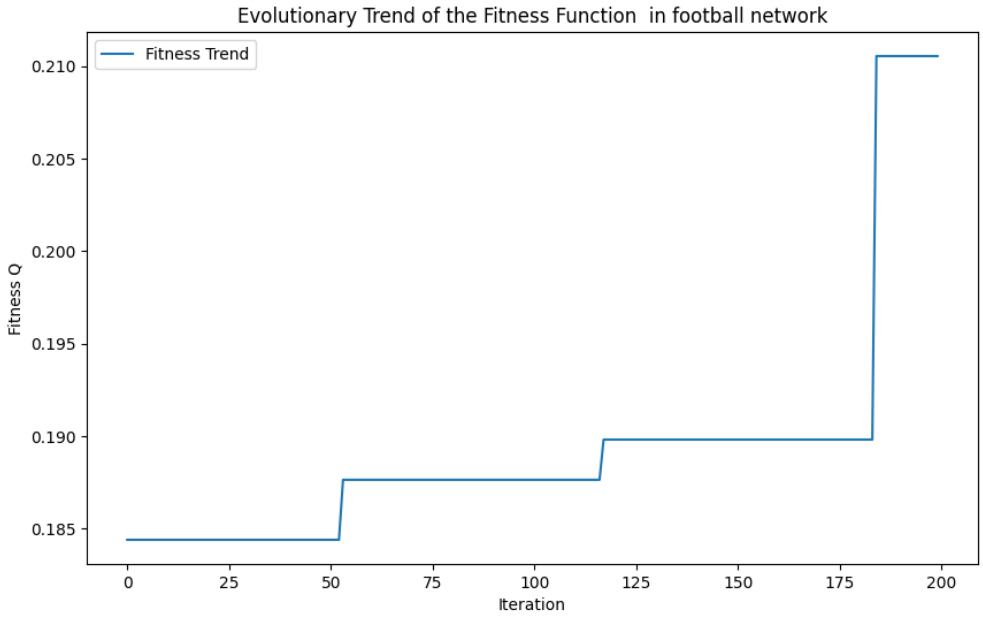


Figure 5.25: Evolutionary trend of IDPSO-RO algorithm on football network

Figures (5.24, 5.25) depicts the convergence of IDPSO-RO and MIDPSO-RO algorithms on football network. As can be seen from the evolutionary trend, MIDPSO-RO algorithm converges faster because Fitness D_λ reaches a stable value earlier (between 125 and 150 iterations) compared to fitness Q (around 175 iterations).

As can be seen from the evolutionary trend, MIDPSO-RO algorithm converges faster in three network(dolphin,polbooks,football). MIDPSO-RO algorithm converges relatively slower but can find a good solution in karate network. local search strategy and Majority Voting strategy makes the algorithm get very close to the real solution, but is also partly responsible for the slower convergence and not very high modularity of MIDPSO-RO algorithm in Karate network. While due to the random factors in updating the individuals' positions, MIDPSO-RO algorithm can get realistic results, and also can search the solution space better to get the best fitness value.

General conclusion

Community detection is still in an exploratory phase and is expected to take several more years to reach maturity. This relative youth of the field has posed challenges, but it has also served as a significant source of motivation.

Detecting community structure is fundamental for the study of complex networks, and community detection can be regarded as a discrete optimization problem. To address this, our research enhances discrete particle swarm optimization algorithms by incorporating a local search strategy and Majority Voting to effectively detect communities. Experimental results demonstrate that the MIDPSO-RO algorithm, utilizing these enhancements, perform well on real networks. In terms of detecting results, the MIDPSO-RO algorithm identifies communities with well modularity. Regarding convergence, the MIDPSO-RO algorithm converges a little faster than the original IDPSO-RO algorithm and is capable of efficiently searching for the optimal solution within the defined space.

In summary, the discrete particle swarm optimization algorithms link an individual's position to the network structure. Our methods enhance the accuracy of detection results by using a local search strategy and majority voting, without requiring prior knowledge of the number or size of communities. We also demonstrate that our algorithm is more efficient in detecting communities compared to popular methods.

This algorithm is simple and easy to understand, performing well on real-world networks. The partitioning results are very satisfactory, and the communities identified are consistent across multiple runs on the same network, making the method easy to implement. The proposed algorithm is fast, with low time complexity, and generally produces stable results.

In the future, we intend to apply our algorithm to community detection in social networks with a large number of nodes. Unlike simple binary connections, social networks often involve weighted connections. Therefore, we plan to compute these weights and address the community detection problem in weighted networks. Additionally, we aim to develop a new recommendation model for social networks that leverages community structures.

Perspectives

To make our approach even more competitive, certain improvements can be made in the future.

Our first perspective is to optimize the refined operator to eliminate randomness. This will enhance the stability of the detected community structures and more well results.

Our second aim is to extend our approach to support weighted communities.

Another idea would be to make our approach applicable in heterogeneous graphs with different types of nodes and links.

Finally, it is desirable to introduce the notion of overlapping communities. This allows our approach to support fuzzy partitioning where a node can belong to several communities.

Bibliography

[1] Introduction to graphs:. http://www.btechsmartclass.com/data_structures/introduction-to-graphs.html [Accessed: (2024-03-01)].

[2] An undirected graph.Håkan,Terelius https://www.researchgate.net/figure/An-undirected-graph-with-7-nodes-and-7-edges[Accessed: (2024-03-01)].

[3] A directed graph . Håkan,Terelius:https://www.researchgate.net/figure/A-directed-graph-with-7-nodes-and-9-edges[Accessed: (2024-03-01)].

[4] Graph theory, 2021. https://calcworkshop.com/trees-graphs/graph-theory/ [Accessed: (2024-03-01)].

[5] Graphs and subgraphs. http://mathonline.wikidot.com/graphs-and-subgraphs [Accessed: (2024-03-01)].

[6] Null graph. Kemal,toker. 2022 https://www.researchgate.net/figure/Null-graph-with-6-vertices [Accessed: (2024-03-01)].

[7] What is the gist of simple graphs, multigraphs,pseudographs?http://ruficalix.com/gist/what-is-the-gist-of-simple-graphs-multigraphs-and-pseudographs-graph-theory/ [Accessed:(2024-03-01)].

[8] Pawel boguslawski. Figure 3 - modelling and analysing 3d building interiors with the dual half-edge data structure. https://www.researchgate.net/figure/Graphs-a-simple-graph-b-multigraph-c-pseudograph-d-labelled-graph-edges-e$_{f}ig4_{2}65219734$ [Accessed:(2024-03-01)].

[9] John Harris, Jeffry L. Hirst, and Michael Mossinghoff. *Combinatorics and Graph Theory*. Undergraduate Texts in Mathematics. Springer New York, New York, NY, 2008.

[10] Sean Gravelle Andrew Hayes Michelle Homp, Alyssa Seideman. Bipartite graphs and stable matchings. https://mathbooks.unl.edu/graph-bipartite/ [Accessed:(2024-03-01)].

[11] Graph Adjacency and Incidence, Gang Wu https://www.baeldung.com/cs/graph-adjacency-and-incidence [Accessed:(2024-03-01)].

[12] Node degrees. https://www.baeldung.com/cs/node-degrees[Accessed:(2024-03-01)].

[13] R. Balakrishnan and K. Ranganathan. *A Textbook of Graph Theory*. Universitext (Berlin. Print). Springer New York, 2012.

[14] Graph kclique percolation https://pythonhosted.org/trustedanalytics/python$_a pi/graphs/graph-/kclique_p ercolation$[Accessed:(2024-03-05)].

[15] Graph representation https://www.javatpoint.com/graph-representation[Accessed:(2024-03-01)].

[16] Graph partition Problem https://www.researchgate.net/figure/Graph-partition-Problem-1-Partitioning-of-a-dynamic-graph-G-V-E-into-k-number[Accessed:(2024-04-01)].

[17] clique clusters https://www.researchgate.net/figure/[Accessed:(2024-04-01)].

[18] Optimum local and global. Yassine,Meraihi https://www.researchgate.net/figure/Optim-local-et-global[Accessed:(2024-03-09)].

[19] A comprehensive survey on the sine–cosine optimization algorithm. https://link.springer.com/article/10.1007/s10462-022-10277-3[Accessed:(2024-03-09)].

[20] Complexity Classes,Jucemar Monteiro https://www.researchgate.net/figure/Diagram-of-intersection-among-classes[Accessed:(2024-03-09)].

[21] Fanzhen,Liu. A Comprehensive Survey on Community Detection With Deep Learning - Scientific Figure on ResearchGatehttps://www.researchgate.net/figure/Practical-applications[Accessed:(2024-03-16)].

[22] Muhammad Aqib Javed, Muhammad Shahzad Younis, Siddique Latif, Junaid Qadir, and Adeel Baig. Community detection in networks: A multidisciplinary review. *Journal of Network and Computer Applications*, 108:87–111, 2018.

[23] Visualization of the steps of the algorithm. https://www.researchgate.net/figure/Visualiz-of-the-steps-of-the-algorithm[Accessed:(2024-03-16)].

[24] The role of endogenous and exogenous mechanisms in the formation of rd networks. https://www.researchgate.net/figure/Two-representative-examples-of-label-propagation-A-labeled-node-whose-label-is-depicted[Accessed:(2024-03-17)].

[25] Subhashis, Majumder.American football network https://www.researchgate.net/figure/American-college-football-network[Accessed:(2024-03-16)].

[26] Journal of Ambient Intelligence and Humanized Computing. American political books network https://www.researchgate.net/figure/American-political-books-network[Accessed:(2024-03-16)].

[27] Clara Pizzuti. Ga-net: A genetic algorithm for community detection in social networks. In *International conference on parallel problem solving from nature*, pages 1081–1090. Springer, 2008.

[28] Jianwu Li and Yulong Song. Community detection in complex networks using extended compact genetic algorithm. *Soft computing*, 17:925–937, 2013.

[29] Qing Cai, Maoguo Gong, Lijia Ma, Shasha Ruan, Fuyan Yuan, and Licheng Jiao. Greedy discrete particle swarm optimization for large-scale social network clustering. *Information Sciences*, 316:503–516, 2015.

[30] Qing Cai, Maoguo Gong, Bo Shen, Lijia Ma, and Licheng Jiao. Discrete particle swarm optimization for identifying community structures in signed social networks. *Neural Networks*, 58:4–13, 2014.

[31] Ferozuddin Riaz and Khidir Ali. Applications of Graph Theory in Computer Science. pages 142–145, August 2011.

[32] Applications of Graph Theory https://www.geeksforgeeks.org/applications-of-graph-theory/[Accessed: (2024-03-01)].

[33] Basic graph theory. https://www.people.vcu.edu/ gasmerom/MAT131/graphs[Accessed: (2024-03-01)].

[34] C. VASUDEV. *GRAPH THEORY WITH APPLICATION*. NEW AGE INTERNATIONAL (P) LIMITED, PUBLISHERS, NEW DELHI, 2006.

[35] Riley,kench. complete graph | definition example. `https://study.com/academy/lesson/complete-graph-definition-example.html`[Accessed: (2024-03-01)].

[36] Types of Graphs https://www.javatpoint.com/graph-theory-types-of-graphs[Accessed: (2024-03-01)].

[37] Pseudograph: Introduction, Definition, Importance:https://testbook.com/maths/pseudograph-Introduction-Definition-Importance[Accessed:(2024-03-03)].

[38] Bipartite Graph:https://mathworld.wolfram.com/BipartiteGraph.htm [Accessed:(2024-03-03)].

[39] Introduction to graph theory. `https://www.whitman.edu/mathematics/cgt_online/book/section04.04.html`.

[40] Robin J. Wilson. *Introduction to graph theory*. Prentice Hall, Harlow Munich, 4. ed., [nachdr.] edition, 2009.

[41] Md Rahman. *Basic Graph Theory*. January 2017.

[42] Nassira Lograda. *La détection de communautés dans les réseaux sociaux*. Thesis, UNIVERSITE MOHAMED BOUDIAF - M'SILA FACULTE DES MATHEMATIQUES ET DE L'INFORMATIQUE DEPARTEMENT D'INFORMATIQUE- Spécialité : Informatique Décisionnel et Optimisation, 2019. Accepted: 2019-07-24T12:50:34Z.

[43] Merazka Mustapha. *Détection de communautés dans les réseaux sociaux*. Thesis, Université Abderahmane MIRA de Bejaia, 2014.

[44] Michael J Barber and John W Clark. Detecting network communities by propagating labels under constraints. *Physical review E*, 80(2):026129, 2009.

[45] Imane Messaoudi. *Détection de communautés dans les réseaux sociaux*. PhD thesis, 2021.

[46] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579, 1901.

[47] Raphaël Tackx. *Analyse de la structure communautaire des réseaux bipartis*. PhD thesis, Sorbonne université, 2018.

[48] Junlong Zhang and Yu Luo. Degree centrality, betweenness centrality, and closeness centrality in social network. In *2017 2nd international conference on modelling, simulation and applied mathematics (MSAM2017)*, pages 300–303. Atlantis press, 2017.

[49] Linton C Freeman et al. Centrality in social networks: Conceptual clarification. *Social network: critical concepts in sociology. Londres: Routledge*, 1:238–263, 2002.

[50] Safaa Belloussaief. *Désintégration d'un réseau social au sein de ses communautés*. PhD thesis, Université du Québec en Outaouais, 2020.

[51] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[52] Khadidja Henni, Neila Mezghani, and Charles Gouin-Vallerand. Unsupervised graph-based feature selection via subspace and pagerank centrality. *Expert Systems with Applications*, 114:46–53, 2018.

[53] Vince Grolmusz. A note on the pagerank of undirected graphs. *Information Processing Letters*, 115(6-8):633–634, 2015.

[54] Adjacency list meaning definition in dsa. https://www.geeksforgeeks.org/adjacency-list [Accessed:(2024-03-03)].

[55] Graph,Representation:https://courses.csail.mit.edu/6.006/spring11/exams/notes2 [Accessed:(2024-03-03)].

[56] Konstantin Andreev and Harald Räcke. Balanced graph partitioning. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 120–124, 2004.

[57] Introduction to Graph Partitioning https://cs.stanford.edu/people/mmahoney/cs369m-Graph-Partitioning[Accessed:(2024-04-01)].

[58] Eugene L Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 2001.

[59] Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[60] Panos M Pardalos and Mauricio GC Resende. Handbook of applied optimization. 2002.

[61] Sara Bouchareb. *Sieve Optimization Method A Survey and Applications*. Thesis, Faculté des Mathématiques et de l'Informatique - Université Mohamed BOUDIAF - M'sila, 2017. Accepted: 2018-01-30T13:16:12Z.

[62] TAIBI Salah and Eddine. *Experimental evaluation of a modified carousel algorithm For the minimum weight vertex cover problem*. Thesis, Faculté des Mathématiques et de l'Informatique - Université Mohamed BOUDIAF - M'sila, 2017. Accepted: 2018-02-04T08:37:04Z.

[63] Optimization models. https://old.mu.ac.in/wp-content/uploads/2017/10/Optimization-Models[Accessed:(2024-03-09)].

[64] F. Rothlauf. *Design of Modern Heuristics: Principles and Application*. Natural Computing Series. Springer Berlin Heidelberg, 2011.

[65] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[66] Hemmak,allaoua.Support de cours d'optimisation combinatoire https://elearning.univ-msila.dz/moodle/pluginfile [Accessed:(2024-03-09)].

[67] Lecture 1 - introduction to optimization. https://www.scribd.com/Lecture-1-Introduction-to-optimization [Accessed:(2024-03-09)].

[68] Mathematical programming ,versus, constraint programming: https://ibm-decision-optimization.github.io [Accessed:(2024-03-09)].

[69] Xin-She Yang. *Engineering optimization: an introduction with metaheuristic applications*. John Wiley & Sons, 2010.

[70] Professor Aaron Sidford, Discrete Mathematics and Algorithms https://web.stanford.edu/class/cme305 [Accessed: 2024-03-09].

[71] Complexity Classes:https://www.geeksforgeeks.org/types-of-complexity-classes [Accessed: 2024-03-11].

[72] Harry R Lewis. Michael r. πgarey and david s. johnson. computers and intractability. a guide to the theory of np-completeness. wh freeman and company, san francisco1979, x+ 338 pp. *The Journal of Symbolic Logic*, 48(2):498–500, 1983.

[73] Complexity classes. https://cse.buffalo.edu/ regan/papers[Accessed: 2024-03-11].

[74] Ajin,Sunny.complexity,classes:https://medium.com/@ajin.sunny/complexity-classes[Accessed: 2024-03-11].

[75] Deffas Zineb and Kholladi Mohamed Khireddine. Métaheuristiques parallèles À solution unique pour la résolution du problème du q3ap sur grille de calcul. Master's thesis, Université Larbi Ben M'hidi - Om-el-bouaghi, 0.

[76] Salim Bouamama. *Design of a learning method for automatic data extraction*. Thesis, October 2014. Accepted: 2014-10-27T09:59:36Z.

[77] Branch and Bound Algorithm https://www.geeksforgeeks.org/branch-and-bound-algorithm/[Accessed:(2024-03-11)].

[78] What is dynamic programming? working, algorithms, and examples. https://www.spiceworks.com/tech/devops/articles/what-is-dynamic-programming/ [Accessed:(2024-03-11)].

[79] Fatiha Souam, Ali Aïtelhadj, and Riadh Baba-Ali. Dual modularity optimization for detecting overlapping communities in bipartite networks. *Knowledge and information systems*, 40:455–488, 2014.

[80] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, February 2010.

[81] NEDIOUI MED ABDELHAMID. *Fouille et apprentissage automatique dans les réseaux sociaux dynamique*. Mém. de mast, 2015.

[82] Nadia Chouchani. *Une approche de détection des communautés d'intérêt dans les réseaux sociaux : application à la génération d'IHM personnalisées*. Thesis, Université Polytechnique des Hauts-de-France, 2018.

[83] Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *Physics Reports*, 659:1–44, November 2016. arXiv:1608.00163 [physics].

[84] Rachid Djerbi. *Détection de communautés dans les réseaux sociaux*. Thesis, University M'Hamed Bougara of Boumerdes, 2021.

[85] Bolin Chen, Weiwei Fan, Juan Liu, and Fang-Xiang Wu. Identifying protein complexes and functional modules—from static ppi networks to dynamic ppi networks. *Briefings in bioinformatics*, 15(2):177–194, 2014.

[86] Arzum Karatas and Serap Sahin. Application Areas of Community Detection: A Review. In *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*, pages 65–70, Ankara, Turkey, December 2018. IEEE.

[87] Ewa Niewiadomska-Szynkiewicz. Application of social network analysis to the investigation of interpersonal connections. *Journal of Telecommunications and Information Technology*, 2012:81–89, 06 2012.

[88] Arzum Karataş and Serap Şahin. A review on social bot detection techniques and research directions. In *Proc. Int. Security and Cryptology Conference Turkey*, pages 156–161, 2017.

[89] Sabrine Ben Abdrabbah, Raouia Ayachi, and Nahla Ben Amor. Collaborative filtering based on dynamic community detection. *Dynamic Networks and Knowledge Discovery*, 85, 2014.

[90] Deepika Lalwani, Durvasula VLN Somayajulu, and P Radha Krishna. A community driven social recommendation system. In *2015 IEEE International conference on big data (big data)*, pages 821–826. IEEE, 2015.

[91] Muhammad Aqib Javed, Muhammad Shahzad Younis, Siddique Latif, Junaid Qadir, and Adeel Baig. Community detection in networks: A multidisciplinary review. *Journal of Network and Computer Applications*, 108:87–111, 2018.

[92] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[93] J Friedman, T Hastie, and R Tibshirani. The elements of statistical learning.(2001). *NY: Springer Series in Statistics*.

[94] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[95] Gaurav Agarwal and David Kempe. Modularity-maximizing graph communities via mathematical programming. *The European Physical Journal B*, 66:409–418, 2008.

[96] Nelson A Alves. Unveiling community structures in weighted networks. *Physical Review E*, 76(3):036101, 2007.

[97] S Boccaletti, M Ivanchenko, V Latora, A Pluchino, and A Rapisarda. Detecting complex network modularity by dynamical clustering. *Physical Review E*, 75(4):045102, 2007.

[98] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *nature*, 435(7043):814–818, 2005.

[99] Cécile Bothorel, Juan David Cruz, Matteo Magnani, and Barbora Micenkova. Clustering attributed graphs: models, measures and methods. *Network Science*, 3(3):408–444, 2015.

[100] Olivier Gach. *Algorithmes mémétiques de détection de communautés dans les réseaux complexes: techniques palliatives de la limite de résolution*. PhD thesis, Université du Maine, 2013.

[101] Girvan-Newman,algorithm:https://memgraph.github.io/networkx-guide-algorithms-/community-detection/girvan-newman/[Accessed:(2024-03-17)].

[102] Community detection algorithms. https://towardsdatascience.com/community-detection-algorithms[Accessed:(2024-03-17)].

[103] Maël Canu. *Détection de communautés orientée sommet pour des réseaux mobiles opportunistes sociaux*. PhD thesis, Université Pierre et Marie Curie-Paris VI, 2017.

[104] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.

[105] Noor elHouda Louafi. Analyse des réseaux sociaux et détection de communautés. Working Paper, 2019. Accepted: 2019-09-30T13:42:41Z.

[106] Label propagation. https://neo4j.com/docs/graph-data-science/current/algorithms/label-propagation/[Accessed:(2024-03-17)][Accessed:(2024-03-17)].

[107] Huan Li, Ruisheng Zhang, Zhili Zhao, and Xin Liu. Lpa-mni: an improved label propagation algorithm based on modularity and node importance for community detection. *Entropy*, 23(5):497, 2021.

[108] Yan Yuan, Bolun Chen, Yongtao Yu, and Ying Jin. An influence maximisation algorithm based on community detection. *International Journal of Computational Science and Engineering*, 22(1):1–14, 2020.

[109] Leon Danon, Albert Diaz-Guilera, Jordi Duch, and Alex Arenas. Comparing community structure identification. *Journal of statistical mechanics: Theory and experiment*, 2005(09):P09008, 2005.

[110] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

[111] David Lusseau, Karsten Schneider, Oliver J Boisseau, Patti Haase, Elisabeth Slooten, and Steve M Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations: can geographic isolation explain this unique trait? *Behavioral Ecology and Sociobiology*, 54:396–405, 2003.

[112] Juyong Park and Mark EJ Newman. A network-based ranking system for us college football. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(10):P10014, 2005.

[113] Valdis Krebs. Proxy networks. analyzing one network to reveal another. *Bulletin de méthodologie sociologique. Bulletin of sociological methodology*, (79):61–70, 2003.

[114] Cen Cao, Qingjian Ni, and Yuqing Zhai. A novel community detection method based on discrete particle swarm optimization algorithms in complex networks. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 171–178, 2015.

[115] Jure Leskovec, Kevin J Lang, and Michael Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th international conference on World wide web*, pages 631–640, 2010.

[116] Pradip Kumar Sahu, Kanchan Manna, Nisarg Shah, and Santanu Chattopadhyay. Extending kernighan–lin partitioning heuristic for application mapping onto network-on-chip. *Journal of Systems Architecture*, 60(7):562–578, 2014.

[117] Aykut Firat, Sangit Chatterjee, and Mustafa Yilmaz. Genetic clustering of social networks using random walks. *Computational Statistics & Data Analysis*, 51(12):6285–6294, 2007.

[118] Clara Pizzuti. A multi-objective genetic algorithm for community detection in networks. In *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, pages 379–386. IEEE, 2009.

[119] Yuzhong Chen and Xiaohui Qiu. Detecting community structures in social networks with particle swarm optimization. In *Frontiers in Internet Technologies: Second CCF Internet Conference of China, ICoC 2013, Zhangjiajie, China, July 10, 2013, Revised Selected Papers*, pages 266–275. Springer, 2013.

[120] F Huang and N Xiao. Particle-swarm-optimization algorithm to discover network community. *Control Theory & Applications*, 28(9):1135–1140, 2011.

[121] Duan Xiaodong, Wang Cunrui, Liu Xiangdong, and Lin Yanping. Web community detection model using particle swarm optimization. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 1074–1079. IEEE, 2008.

[122] Qing Cai, Maoguo Gong, Bo Shen, Lijia Ma, and Licheng Jiao. Discrete particle swarm optimization for identifying community structures in signed social networks. *Neural Networks*, 58:4–13, 2014.

[123] Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the sixth international symposium on micro machine and human science*, pages 39–43. Ieee, 1995.

[124] James Kennedy. Particle Swarm Optimization. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, pages 760–766. Springer US, Boston, MA, 2010.

[125] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization: An overview. *Swarm intelligence*, 1:33–57, 2007.

[126] Zhenping Li, Shihua Zhang, Rui-Sheng Wang, Xiang-Sun Zhang, and Luonan Chen. Quantitative function for community detection. *Physical review E*, 77(3):036109, 2008.

[127] Anping Song, Mingbo Li, Xuehai Ding, Wei Cao, and Ke Pu. Community detection using discrete bat algorithm. *IAENG International Journal of Computer Science*, 43(1):37–43, 2016.

[128] Python Tutorial https://www.tutorialspoint.com/python[Accessed:(2024-05-22)].

[129] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

ملخص

يمكن نمذجة العديد من الأنظمة المعقدة الحقيقية في العالم الحقيقي كشبكات معقدة، والتي تكشف عن خصائص مهمة مثل بنية المجتمع. بنية المجتمع لها العديد من التطبيقات في مجالات علمية مختلفة. في هذا العمل، نقوم بتوضيح التطبيقات المختلفة لبنية المجتمع. بعد ذلك، نقدم تعريفًا لمشكلة بنية المجتمع ونطبق خوارزمية مستوحاة من علم الأحياء للكشف عن بنية المجتمع. سنختبر الخوارزمية في مجموعة بيانات شبكة حقيقية.

الكلمات المفتاحية: بنية المجتمع، نظرية الرسم البياني، التحسين، الكشف عن المجتمع، تحسين سرب الجسيمات، تحسين سرب الجسيمات المنفصل

Abstract

Many real complex system in real world can be modeled as complex networks , which reveal important characteristics such as community structure. The community structure has many applications in different scientific field. In this work, we tray to illustrate the different applications of community structure . Then, we provide the problem definition of the community structure and we applied bio-inspired algorithm to reveal the community structure. We will teste the algorithm in real-network data-set.

Keywords: Community Structure, Community Detection, Graph Theory, Optimization, PSO, Discret PSO

Résumé

De nombreux systèmes complexes réels peuvent être modélisés sous forme de réseaux complexes, qui révèlent des caractéristiques importantes telles que la structure de la communauté. La structure communautaire a de nombreuses applications dans différents domaines scientifiques. Dans ce travail, nous illustrons les différentes applications de la structure communautaire. Ensuite, nous fournissons la définition du problème de la structure de la communauté et nous appliquons un algorithme bio-inspiré pour révéler la structure de la communauté. Nous testerons l'algorithme sur un ensemble de données de réseaux réels.

Mots clés : Structure de la Communauté, Détection de Communautés, Théorie des Graphes, Optimisation, PSO, PSO discrète