PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
SETIF1 UNIVERSITY- FERHAT ABBAS
FACULTY OF SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

Hybrid metaheuristic approach for community detection in complex networks.

A thesis presented by:

TAIBI Salah Eddine

As a requirement to aim for the degree of:

3rd cycle Doctorate in Computer Science

Approved by supervisory committee:

Dr. Semchedine Moussa	Setif1 University-Ferhat Abbas	President
Prof. Bouamama Salim	Setif1 University-Ferhat Abbas	Supervisor
Dr. Toumi Lyazid	Setif1 University-Ferhat Abbas	Co-supervisor
Prof. Nouioua Farid	University of Bordj Bou Arreridj	Examinator
Prof. Hemmak Allaoua	University of M'sila-Mohamed Boudiaf	Examinator
Dr. Balbal Samir	Setif1 University-Ferhat Abbas	Examinator
Dr. Slimani Yacine	Setif1 University-Ferhat Abbas	Guest

2025/2026


بسم الله الرحمن الرحيم

Abstract

Optimization is a scientific field dedicated to identifying optimal solutions from a set of feasible alternatives. Numerous critical problems in business, economics, and engineering can be formulated as optimization tasks. However, most such problems are classified as NP-Hard, meaning that finding an exact optimal solution is computationally infeasible for large-scale instances. In such cases, stochastic meta-heuristic methods are preferred as they provide near-optimal solutions within a reasonable timeframe. This stands in contrast to exact methods, which guarantee optimality but at the cost of exponentially longer runtimes. Consequently, the hybridization of optimization methods has garnered significant research interest in recent years as a strategy to reduce computational time while enhancing solution quality, i.e., both effectiveness and accuracy.

This thesis addresses a specific NP-Hard optimization problem: community detection in complex networks, where the objective is to maximize the modularity metric. To tackle this challenge, we propose two novel hybrid meta-heuristics:

- The Clustering Coefficient Discrete Bat Algorithm (CC-DBAT): This approach utilizes the clustering coefficient to generate an intelligent initial population. It is further enhanced by hybridizing it with the Fast Local Move heuristic and a random neighbors technique to improve its search capability.
- The Fast Local Move Iterated Greedy (FLMIG) Algorithm: This method hybridizes the Iterated Greedy (IG) framework with the Fast Local Move heuristic. A novel reconstruction procedure is incorporated to ensure the connectivity of the detected communities.

Experimental results, evaluated using multiple performance metrics, demonstrate that the proposed algorithms outperform existing state-of-the-art methods, achieving superior performance in both solution quality and computational efficiency.

Keywords: Hybrid metaheuristic, Social network analysis, Community discovery, Modularity maximization

Résumé

L'optimisation est un domaine scientifique dédié à l'identification de solutions optimales parmi un ensemble d'alternatives réalisables. De nombreux problèmes critiques en entreprise, économie et ingénierie peuvent être formulés comme des tâches d'optimisation. Cependant, la plupart de ces problèmes sont classés comme NP-Difficiles, ce qui signifie qu'il est informatiquement impossible de trouver une solution optimale exacte pour des instances à grande échelle. Dans de tels cas, les méthodes stochastiques méta-heuristiques sont préférées car elles fournissent des solutions quasi optimales dans un délai raisonnable. Cela contraste avec les méthodes exactes, qui garantissent l'optimalité mais au prix de temps d'exécution exponentiellement plus longs. Par conséquent, l'hybridation des méthodes d'optimisation a suscité un intérêt de recherche significatif ces dernières années en tant que stratégie pour réduire le temps de calcul tout en améliorant la qualité des solutions, c'est-à-dire à la fois l'efficacité et la précision.

Cette thèse aborde un problème d'optimisation NP-Difficile spécifique : la détection de communautés dans les réseaux complexes, où l'objectif est de maximiser la métrique de modularité. Pour relever ce défi, nous proposons deux nouvelles méta-heuristiques hybrides :

- L'algorithme Clustering Coefficient Discrete Bat (CC-DBAT) : Cette approche utilise le coefficient de clustering pour générer une population initiale intelligente. Elle est en outre améliorée par son hybridation avec l'heuristique Fast Local Move et une technique de voisins aléatoires pour accroître ses capacités d'exploration.
- L'algorithme Fast Local Move Iterated Greedy (FLMIG) : Cette méthode hybride le framework Iterated Greedy (IG) avec l'heuristique Fast Local Move. Une nouvelle procédure de reconstruction est intégrée pour garantir la connectivité des communautés détectées.

Les résultats expérimentaux, évalués à l'aide de multiples métriques de performance, démontrent que les algorithmes proposés surpassent les méthodes existantes de l'état de l'art, atteignant des performances supérieures tant en qualité de solution qu'en efficacité computationnelle.

Keywords: Méta-heuristique hybride, Analyse de réseaux sociaux, Détection de communautés, Maximisation de la modularité

الملخص

التحسين التوافقي هو مجال علمي مخصص لتحديد الحلول المثلى من مجموعة من البدائل الممكنة. يمكن صياغة العديد من المشكلات الحرجة في مجالات الأعمال والاقتصاد والهندسة كمهام تحسين. ومع ذلك، فإن معظم هذه المشكلات تُصنف على أنها NP-Hard، مما يعني أن إيجاد الحل الأمثل بدقة يكون غير ممكن حسابيًا في حالات النطاق الكبير. في مثل هذه الحالات، يُفضل استخدام لأنها توفر حلولًا شبه مثالية ضمن إطار زمني معقول. وهذا يقفالطرق الاستدلالية العشوائية (stochastic meta-heuristics) على النقيض من الطرق الدقيقة، التي تضمن حلول مثالية ولكن على حساب وقت تشغيل أطول بشكل أسي. ونتيجة لذلك، حظي تهجين خوارزميات التحسين باهتمام بحثي كبير في السنوات الأخيرة كاستراتيجية لتقليل الوقت الحسابي مع تحسين جودة الحل، أي من خلاله يتم تحسين فعالية ودقة أداء الخوارزميات.

تتناول هذه الرسالة مشكلة تعريف المجتمات في الانظمة المعقدة حيث يندرج هذا المشكل في مسائل التحسين التوافقي NP-Hard (modularity) حيث الهدف هو تعظيم معيار التجزئة.

ولمواجهة هذا التحدي، نقترح طريقتين جديدتين من الاستدلاليات الهجينة:

- خوارزمية الخفافيش المنفصلة المهجنة بوساطة معامل التجمع: تستخدم هذه الطريقة معامل التجمع لتوليد(CC-DBAT) مجموعة أولية، ويتم تحسين المجوعة الأولية عن طريق تهجين الخوارزمية الاساسية بأسلوبين استدلالين , ألأول التحرك المحلي والثاني تقنية اختيار المجتمعات المجاورة عشوائياالسريع (Fast Local Move)

  : تدمج هذهخوارزمية التحرك المحلي السريع مع الخوارزمية الجشعة (Fast Local Move Iterated Greedy – FLMIG) كما يتم دمج إجراء إعادة بناء جديد لضمان الاتصال الداخلي بينالطريقة الخوارزمية الأساسية (IG) Iterated Greedy عناصر المجتمع الواحد.

تُظهر النتائج التجريبية، التي تم تقييمها باستخدام عدة مقاييس للأداء، أن الخوارزميات المقترحة تتفوق على الطرق الحالية المتقدمة، محققة أداءً أفضل من حيث جودة الحل والكفاءة الحسابية.

Keywords: التحسين التوافقي , مقياس التجزئة, تعريف المجتمعات

Dedication

Dedicated to my mother, and to the cherished memory of my father, my brothers, and my sisters.

ACKNOWLEDGEMENTS

First and foremost, I wish to express my deepest gratitude to my supervisor, Professor Salim Bouamama, and my co-supervisor, Dr. Toumi Lyazid, for their unwavering support, invaluable guidance, and constant encouragement throughout this research. Without their insightful feedback and expert advice, this thesis would not have been possible.

I am also profoundly grateful to the members of my dissertation committee for honoring me by accepting to evaluate my work and for their time and consideration.

Furthermore, I would like to extend my sincere thanks to all the teachers and professors who have instructed and inspired me at various stages of my academic journey.

AUTHOR'S DECLARATION

We declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programs, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such

EMAIL :

SIGNED: ..... DATE: .....

TABLE OF CONTENTS

LIST OF TABLES

TABLE **Page**

List of Figures

List of Algorithms

CHAPTER 1

INTRODUCTION

1.1 Overview:

Complex networks provide a powerful framework for modeling diverse real-world systems, ranging from social interactions and biological processes to communication infrastructures , transportation grids [43], technological ecosystems[118], ecological interdependencies [10], and information flows [36], health domain [145]. These networks permeate our daily existence, offering structural insights into inherently interconnected phenomena.

A main challenge in analyzing such systems is the identification of meaningful substructures (commonly called communities) embedded within the larger network topology [130]. Communities are typified by dense intra-group connections compared with sparser inter-group linkages, reflecting functional or organizational modules. However, the absence of a universal, rigorous definition for what constitutes a "community" has rendered this pursuit intrinsically complex, galvanizing the field known as the Community Detection (CD) problem [55].

Finding community structures has gained attention in recent years due to the fact that it provides important information on the fundamental processes influencing network behaviour. Social networks [123], biology [10], Energy [169], personalised recommendation systems [181], security [168], epidemic modelling [57], and transportation systems [43] are only a few of the domains where this feature is especially crucial. To put it simply, the CD problem is a challenging task that aims to create data analysis methods that can separate large-scale networked datasets into discrete, cohesive clusters where the size and number of these subgroups are unknown beforehand.

Critically, real-world interactions are inherently dynamic: social ties form and dissolve, communication events are transient, and mobility patterns evolve temporally. Despite this reality, predominant CD literature relies on a static network paradigm, implicitly assuming

system quiescence. This simplification often fails to capture temporal heterogeneities, leading to potentially obsolete or inaccurate partitions. Consequently, the integration of temporal dynamics into graph-theoretic methods has emerged as a pivotal research impetus over the past decade, catalyzing advances in dynamic network analysis. Within this paradigm, canonical problems originally formulated for static graphsincluding CDhave been reconceptualized, spurring the development of novel time-aware computational approaches [138].

The growing number of proposed approaches in the literature has been analyzed from various perspectives and strategies. Consequently, several reviews have been conducted to systematically examine and categorize these approaches. These reviews typically classify the methodologies based on the type of graph employed and the specific strategies they utilize. Fortunato [54][55] presented a comprehensive review to associate several heuristic techniques with graph partitioning, with his work being one of the earliest in the field. Subsequently, a series of reviews have been proposed, extensively discussing the problem from various perspectives. These reviews primarily categorize the approaches into heuristic and metaheuristic-based methods.

Building on foundational research in dynamic network analysis, Rosseiti et al. [138] introduced a comprehensive practical guide that systematically organizes Dynamic Community Detection (DCD) algorithms into a principled taxonomy. This framework classifies methodologies according to their underlying computational principles and implementation strategies for identifying temporally coherent substructures in evolving networks. Their proposed taxonomy delineates three primary algorithmic classes, each comprising distinct methodological categoriesa classification we will examine in detail in subsequent sections.

Complementing this structural taxonomy, Dakiche et al. [41] developed an operational classification scheme specifically for community discovery in dynamic social networks. Their framework categorizes approaches based on temporal processing strategies: (1) methods employing independent static detection followed by cross-temporal matching; (2) techniques using dependent sequential static detection; (3) approaches performing simultaneous analysis across all evolutionary stages; and (4) methodologies operating directly on temporal network.

Extending the analytical scope to multilayer systems, Matteo Magnani et al. [106] established a tripartite classification for community discovery in multiplex networks: flattening (projecting layers into single networks), aggregation (combining layer information), and direct (multilayernative) approaches. Their comprehensive survey examines network modeling formalisms, quality metrics, and algorithmic strategiesincluding label propagation, multi-objective optimization, and embedding techniques such as random walks and non-negative matrix factorization. Critically, they identified persistent research gaps: the absence of universally efficient strategies for large-scale multiplex partitioning, and the lack of established methodologies for developing domain-agnostic algorithms.

Community detection (CD) constitutes a combinatorial optimization problem formally classified as NP-hard [54][27][26], establishing its inherent computational complexity. Optimization-

based CD methods are mostly categorized by strategic paradigm into two principal classes: Heuristic Community Detection (HCD) algorithms, leveraging problem-specific rules, and Metaheuristic Community Detection (MCD) algorithms, employing high-level strategies for exploring complex solution landscapes.

Several fundamental reviews have systematically analyzed evolutionary computation approaches for CD. Cai et al. [31] surveyed evolutionary frameworks for both single and multiobjective CD, categorizing methods according to underlying network structural properties. Complementing this, Pizzuti [129] provided a comprehensive examination of state-of-the-art evolutionary and bio-inspired CD algorithms. This review exactly details implementations of core componentsincluding solution representation, fitness evaluation, and evolutionary operatorsacross diverse network typologies: directed/undirected, weighted/signed, multidimensional, overlapping, and temporal. While [31] typically explores evolutionary optimization, [129] specifically bridges methodological design with CD applications in complex network analysis.

Osaba et al. [121] conducted a critical evaluation of modern MCD algorithms, emphasizing emerging bio-inspired metaheuristics. Their work empirically assessed seven prominent techniques on weighted directed networks and introduced two novel perturbation operators: "blind movement" and "ad-hoc heuristic." Results revealed that integrating simplistic heuristic mechanisms yielded inconclusive performance gains, with no single algorithm demonstrating clear superiority. Baraa A. Attea et al. [11] recently advanced the field through an integrative taxonomy, classifying optimization-based CD methodologies into four strategic categories: Heuristics (HCD), Meta-Heuristics (MCD), Hybrid-Metaheuristics, and Hyper-Metaheuristics, the latter two conceptualized as foundational frameworks for next-generation algorithm design. Their analysis further encompassed diverse network models, including undirected, directed, weighted, signed, multiplex, overlapping, and temporal networks.

The effectiveness of community detection algorithms is typically evaluated using several metrics. Two of the most prominent are Normalized Mutual Information (NMI) [42] and modularity [116]. NMI quantifies the similarity between a detected partition and a ground-truth reference, whereas modularity measures the quality of a partition by comparing the density of edges within communities to a random network model.

Despite extensive research, the community detection problem remains a significant challenge. Many existing approaches suffer from poor scalability on large, complex networks or focus solely on maximizing modularity without sufficiently ensuring detection accuracy.

This thesis addresses these limitations by proposing novel hybrid meta-heuristic methodologies, introducing a two-pronged approach:

- Clustering Coefficient-based Discrete Bat Algorithm (CC-DBAT): An enhanced discrete bat algorithm that uses the clustering coefficient to identify communities. This method is hybridized with the Fast Local Move technique [122] and a random neighbors strategy [162] to improve its effectiveness.

- Fast Local Move Iterated Greedy Algorithm (FLMIG): A novel metaheuristic that extends the Louvain Prune heuristic. It integrates the Fast Local Move (FLM) method [122] within the Iterated Greedy (IG) framework [66, 143] to maximize modularity for community detection.

Furthermore, this thesis proposes a novel taxonomy for categorizing optimization-based community detection methods according to their underlying strategies and examines their evolution in response to different network models. Most significantly, it pioneers a new research direction by integrating network embedding techniques into meta-heuristic frameworks, an approach applicable to both static and dynamic networks. The practical implications of this research are vast, spanning numerous domains such as energy [169], personalized recommendation systems [123], security [183], epidemic modeling [57], transportation systems [120], biological networks [10], and social network analysis [181].

1.2 Our contributions:

The primary goal of this thesis is to create novel community detection algorithms that maximize modularity and to introduce a new taxonomy for optimization-based methods. These methods target high-quality solutions within feasible computational time. The main contributions are:

1. **An Enhanced Bat Algorithm Using Clustering Coefficient for Community Detection in Complex Networks:** This study introduces CC-DBAT, an enhanced Discrete Bat Algorithm utilizing the Clustering Coefficient that improves upon DBAT-M [2]. Our approach strengthens local search and randomization for more comprehensive search space exploration by: (1) initializing populations via clustering coefficient [144] to establish weak inter-community connections and high-quality starting structures; (2) refining solutions through DBAT hybridization with fast local moves [122] (replacing standard local search) to optimize exploration-exploitation; and (3) incorporating random community neighbors [162] to balance diversification-intensification. Experimental validation on synthetic and real-world networks demonstrates CC-DBAT's superior performance against existing bat algorithms across multiple criteria: higher modularity and NMI scores, faster execution times, and improved convergence rates.

2. **Complex Network Community Discovery Using Fast Local Move Iterated Greedy Algorithm:** In this study, we propose FLMIG (Fast Local Move Iterated Greedy), a novel algorithm that enhances the Louvain Prune heuristic through an Iterated Greedy framework for modularity optimization in non-overlapping communities. FLMIG integrates: (i) efficient local optimization via fast local moves, (ii) iterative enhancement through destruction-reconstruction cycles, and (iii) a refinement phase ensuring internally connected communities. This integrated approach overcomes the connectivity limitations of

prior methods while maintaining computational efficiency through strategic local search operations. We conducted experiments on real-world and synthetic datasets to evaluate the efficiency of the Fast Local Move Iterated Greedy (FLMIG) algorithm against Leiden (LDN) [163], Louvain Prune (LVNP) [122], and Iterated Carousel Greedy (ICG) [84]. The algorithm demonstrates comparable or superior performance to state-of-the-art methods like Leiden and Louvain Prune, especially in large-scale networks.

1.3 Thesis organization:

This thesis is structured as follows:

Chapter 2: Outlines the thesis background, defining key network models, preliminary notation, and the core optimization problem along with its classifications. It also provides a comprehensive overview of resolution methods, heuristics, and metaheuristics used to solve such problems.

Chapter 3: Focuses on the community detection problem. We first formally define the problem within network models. Second, we present metrics for assessing detected community quality. Third, we propose a novel taxonomy classifying optimization-based methods according to their behavioral characteristics, analyzing the advantages and limitations of each class. Finally, we discuss recent advances within each category.

Chapters 4, 5: Each subsequent chapter details one of the two core contributions developed in this work. Specifically, Chapter 4 describes the framework for the CC-DBAT approach, while Chapter 5 details the FLMIG framework. Both chapters include pseudocode for primary algorithmic components and provide comprehensive analysis of experimental results.

Chapter 6: Concludes the thesis and outlines future research directions.

1.4 Academic Publications Produced:

The following academic paper have been produced as result of this reaserch :

- S. TAIBI, S. BOUAMAMA, AND L. TOUMI, *An enhanced bat algorithm using clustering coefficient for community detection in complex networks*, in 2024 1st International Conference on Innovative and Intelligent Information Technologies (IC3IT), IEEE, 2024, pp. 1–6.

- S. TAIBI, L. TOUMI, AND S. BOUAMAMA, *Complex network community discovery using fast local move iterated greedy algorithm*, The Journal of Supercomputing, 81 (2025), p. 182.

CHAPTER 2

BACKGROUND

"Graph theory teaches us that connections matter more than individual points."

2.1 Introduction

This chapter provides a foundational overview of graph theory, presenting various graph types and essential notation crucial for modeling complex systems. It then delivers a broad introduction to combinatorial optimization, covering its core definition, a taxonomy of problem types, and an examination of computational complexity. The chapter concludes by presenting a taxonomy of optimization methods.

2.2 Complex networks topology

Complex networks can be classified based on their structural properties. The most fundamental type is the unsigned, undirected, and unweighted graph, where even core tasks like community detection (CD) are computationally challenging (NP-hard). Real-world networks, however, are often more nuanced. They frequently contain directed links that can be signed (denoting positive/negative relationships) or weighted (reflecting connection strength), capturing the complexity of real-world interactions.

To model multiple types of interactions within the same set of entities, multiplex networks are used. This framework is applicable to diverse systems, from biological organisms and social structures to transportation infrastructure. For example, a biological interactomea network of

thousands of protein moleculesmay involve distinct interaction mechanisms representing various functional relationships [106]. A classic real-world example is a social system where the same group of people interact on different platforms or in different contexts.

Initially, research focused on narrowly defined mining tasks for static networks, treated as single "snapshots" in time. Foundational work on problems like community detection [54], node ranking [48], and frequent pattern mining [3] was developed under this static paradigm, which fails to capture the inherent dynamism of real systems.

A common approach to modeling dynamics uses a sequence of time-ordered network snapshots. While this method effectively tracks structural evolution and offers flexibility, it increases analytical complexity. For time-sensitive tasks, this introduces two primary challenges: (i) effectively tracking the network's evolving phases, and (ii) ensuring consistency by reconciling insights from one snapshot with those from subsequent ones.

Recent studies advocate for retaining full temporal granularity instead of aggregating data into snapshots, a paradigm known as temporal networks. These models represent a dynamic network as a chronological sequence of timestamped interactions [138].

The complexity of network analysis stems from four key elements: data topology, node attributes, relationship dynamics, and network type. These components, individually or in combination, create unique analytical challenges that shape the design of detection methodologies. We classify complex network types and formalize their mathematical models accordingly, as shown in Fig 2.7.

2.2.1 Undirected Networks:

A complex network is modeled as a *simple undirected graph* $\mathsf{G} = (\mathsf{V}, \mathsf{E})$, where:

- $\mathsf{V} = \{1, \ldots, n\}$ is the vertex set ($|\mathsf{V}| = n$)
- $\mathsf{E} \subseteq \big\{\{u, v\} \mid u, v \in \mathsf{V}\big\}$ is the edge set ($|\mathsf{E}| = m$)
- An edge exists between vertices u and v if and only if $\{u, v\} \in E$

The graph is represented by its *adjacency matrix* $A \in \{0,1\}^{n \times n}$:

$$A_{uv} = \begin{cases} 1 & \text{if } \{u, v\} \in \mathsf{E} \\ 0 & \text{otherwise} \end{cases}$$

For any vertex $u \in \mathsf{V}$:

- *Neighborhood*: $N_u = \{v \mid \{u, v\} \in E\}$
- *Degree*: $d_u = |N_u| = \sum_{v=1}^{n} A_{uv}$

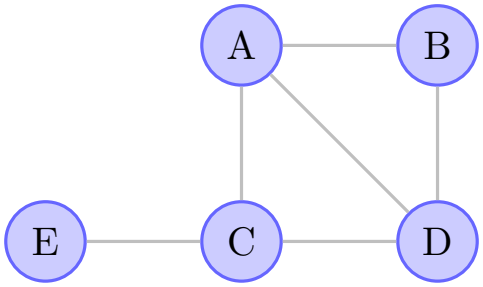


Figure 2.1: A simple undirected graph.

2.2.2 Directed Networks:

A complex network is modeled as a *directed graph* $\mathsf{G} = (\mathsf{V}, \mathsf{E})$, where:

- $\mathsf{V} = \{1, \ldots, n\}$ is the vertex set ($|\mathsf{V}| = n$)
- $\mathsf{E} \subseteq \mathsf{V} \times \mathsf{V}$ is the set of directed edges ($|\mathsf{E}| = m$)
- A directed edge from vertex u to v exists if $(u, v) \in \mathsf{E}$

The graph is represented by its *asymmetric adjacency matrix* $A \in \{0,1\}^{n \times n}$:

$$A_{uv} = \begin{cases} 1 & \text{if } (u,v) \in \mathsf{E} \\ 0 & \text{otherwise} \end{cases}$$

For any vertex $u \in \mathsf{V}$:

- *Out-degree*: $d_u^{\text{out}} = \sum_{v=1}^{n} A_{uv}$ (sum of row u)
- *In-degree*: $d_u^{\text{in}} = \sum_{v=1}^{n} A_{vu}$ (sum of column u)

The total edge count satisfies:

$$m = \sum_{u=1}^{n} d_u^{\text{out}} = \sum_{u=1}^{n} d_u^{\text{in}}$$

2.2.3 Weighted Networks:

A complex network is modeled as a *weighted graph* $G = (V, E, W)$, where:

- $V = \{1, \ldots, n\}$ is the vertex set ($|V| = n$)
- $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$ is the edge set ($|E| = m$)
- $W : E \to \mathbb{R}$ is a weight function assigning real values to edges

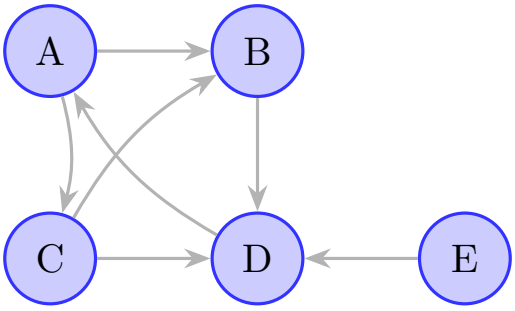


Figure 2.2: A simple directed graph.

The graph is represented by its *weighted adjacency matrix* $A \in \mathbb{R}^{n \times n}$:

$$A_{uv} = \begin{cases} w_{uv} & \text{if } \{u,v\} \in E \\ 0 & \text{otherwise} \end{cases}$$

where $A_{uv} > 0$ indicates an edge between u and v with weight w_{uv}.

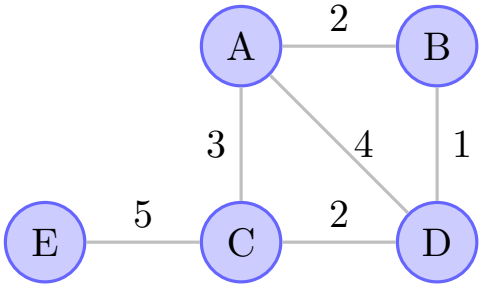


Figure 2.3: A weighted graph.

2.2.4 Signed Networks:

A complex network is modeled as a *signed graph* $G = (V, E)$, where:

- $V = \{1, \ldots, n\}$ is the vertex set ($|V| = n$)
- $E \subseteq \{\{u,v\} \mid u,v \in V\}$ is the edge set ($|E| = m$)
- An edge $\{u,v\} \in E$ represents a signed connection between u and v

The graph is represented by its *signed adjacency matrix* $A \in \{-1,0,1\}^{n\times n}$:

$$A_{uv} = \begin{cases} 1 & \text{if positive edge } \{u,v\} \in E \\ -1 & \text{if negative edge } \{u,v\} \in E \\ 0 & \text{otherwise} \end{cases}$$

For any vertex $u \in V$:

- *Positive neighbors*: $N_u^+ = \{v \mid \{u,v\} \in E \text{ and } A_{uv} = 1\}$
- *Positive degree*: $d_u^+ = |N_u^+|$
- *Negative neighbors*: $N_u^- = \{v \mid \{u,v\} \in E \text{ and } A_{uv} = -1\}$
- *Negative degree*: $d_u^- = |N_u^-|$
- *Total degree*: $d_u = d_u^+ + d_u^-$

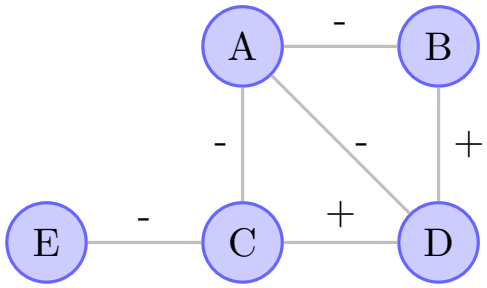


Figure 2.4: A signed graph.

2.2.5 Attributed network:

An attributed network can be defined as a non-directed graph $G = (A,X)$, where $A \in R^{n\times n}$ denotes the adjacency matrix with n nodes, $X \in R^{n\times d}$ indicates the attribute matrix with d features.

2.2.6 Multiplex Networks:

A complex system is modeled as a *multilayer network* [80] $G_M = (V_M, \mathbb{E}, V, \mathscr{L})$, where:

- $V = \{1,\ldots,n\}$ is the node set
- $\mathscr{L} = \{\mathscr{L}_a\}_{a=1}^d$ is the set of elementary layer sets
- $V_M \subseteq V \times \mathscr{L}_1 \times \cdots \times \mathscr{L}_d$ is the node-layer set, containing tuples $(i, l_1, \ldots, l_d)$ where node i exists in layer combination $(l_1, \ldots, l_d)$

- $\mathbb{E} \subseteq V_M \times V_M$ is the edge set connecting node-layer pairs

Multiplex networks are a special case of multilayer networks [80] where:

- Each layer $l \in \{1,\ldots,d\}$ has identical node set V
- $G_M = \{G_l \mid l = 1,\ldots,d\}$ where $G_l = (V, E_l)$ is the l^{th} layer graph
- Edges are strictly *intra-layer*: $E_l \subseteq \{\{(u,l),(v,l)\} \mid u,v \in V\}$
- The global edge set $\mathbb{E} = \bigcup_{l=1}^{d} E_l$ with $((i,l_1),(j,l_2)) \in \mathbb{E}$ only if $l_1 = l_2$

Each layer G_l may be directed/undirected and weighted/unweighted, but shares node set V across layers.

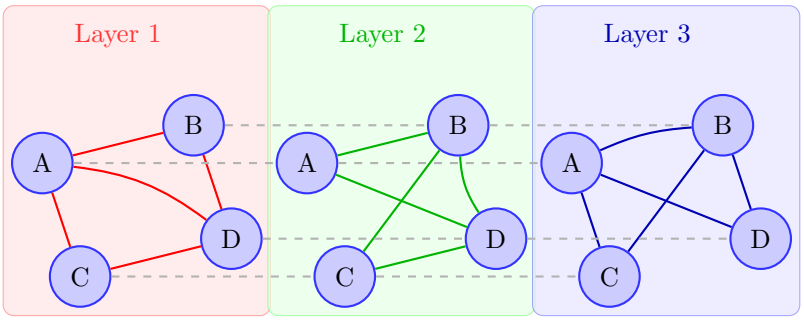


Figure 2.5: A multiplex graph.

2.2.7 Dynamic Networks:

2.2.7.1 Temporal Networks:

The dynamic structure of complex networks, which the state of nodes and edges vary with time, can be modeled as Temporal Networks. The temporal networks can be represented as a Temporal Network is a graph $G_t = (V_T, W, E_T, T)$ where: V_T is a set of nodes and W is the set of temporal nodes $W \subseteq V_T \times T$, and E_T is the set of edges $E_T \subseteq T \times V_T \otimes V_T$ such that $(t, uv) \in E_T$ implies $(t,u) \in W$ and $(t,v) \in W$ [69].

2.2.7.2 Snapshot Networks:

A temporal network can be represented as a sequence of ordered set [112] $G_t = \{G_1, G_2, G_3, G_4, ...G_k\}$, where each $G_t = (V, E_t)$ is a static graph snapshot at time t. This is common in discrete-time models, k is the number of timestamps of G_t. Depending on the network semantics, we can deal with undirected Snapshot Graphs (SN) as presented in Fig2.6 or with Directed Snapshot Graphs (DSN).

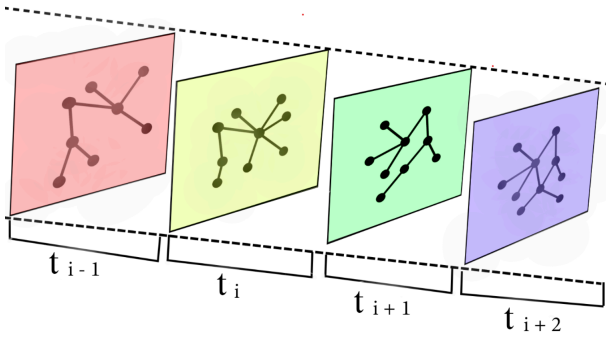


Figure 2.6: Dynamics snapshot network.

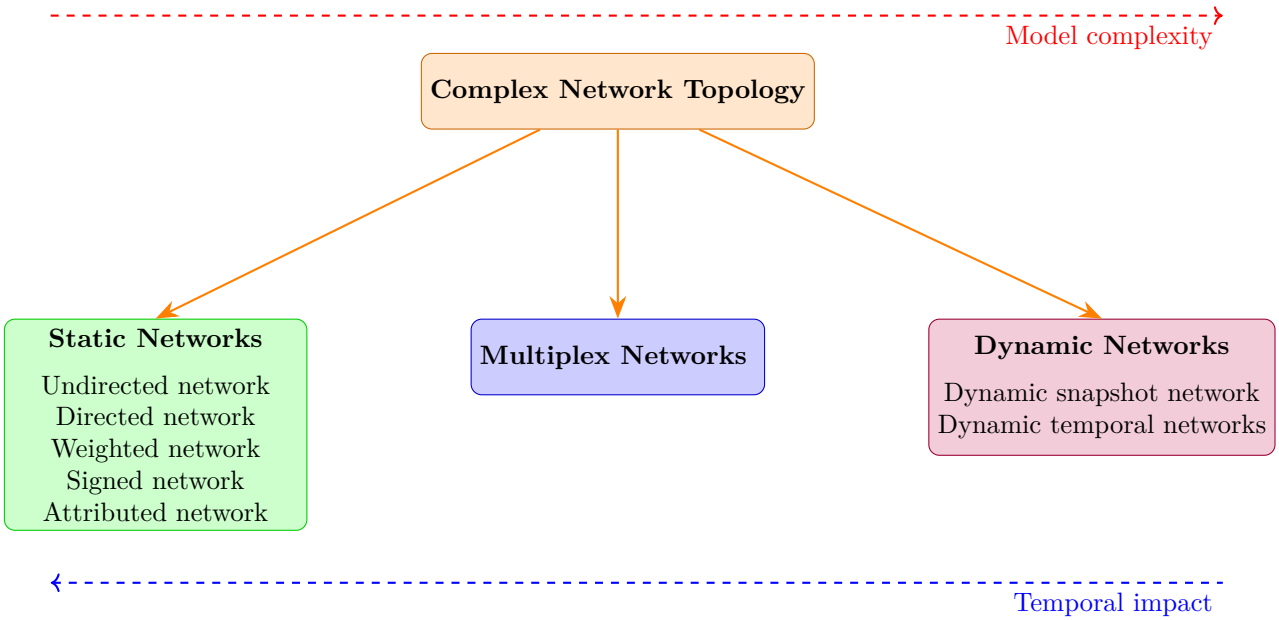


Figure 2.7: **Network Representation Complexity:** Transitioning from static graph representations (directed, signed, weighted) to interaction-based dynamic models increases computational complexity while reducing the temporal resolution of available network information.

2.3 Optimization problem

Optimization is an essential process for decision-making and analyzing real-world systems. An optimization problem can be formally defined in the following generic form [173]:

$$\text{minimize}_{x \in \mathbb{S}} \quad \mathscr{F}_i(x), \quad (i = 1, 2, \ldots, M), \tag{2.1}$$

$$\text{subject to} \quad \Phi_j(x) = 0, \quad (j = 1, 2, \ldots, J), \tag{2.2}$$

$$\Psi_k(x) \leq 0, \quad (k = 1, 2, \ldots, K), \tag{2.3}$$

where $f_i(x), \Phi_j(x)$ and $\Psi_k(x)$ are functions of the design vector

$$x = (x_1, x_2, \ldots, x_n)^T. \tag{2.4}$$

The elements x within the vector x are known as design variables or decision variables. These variables can be real-valued (continuous), discrete, or a combination of both. The functions $\mathscr{F}_i(x)$ (where $i = 1, 2, ..., M$) are termed the objective functions.

When $M = 1$, the problem involves only a single objective. These functions are also sometimes referred to in literature as cost functions or energy functions. The domain defined by the possible values of the decision variables is called the design space or search space, denoted $\mathbb{S}$. Conversely, the space containing the values of the objective functions is known as the solution space or response space.

The expressions $\Phi_j = 0$ represent equality constraints, while $\Psi_k \leq 0$ (or ≥ 0) represent inequality constraints. It's important to note that the objectives can also be framed as maximization problems.

2.4 Taxonomy of optimization problems

Optimization classifications lack standardization, leading to confusing terminology. It concisely introduces key concepts, grouping methods loosely based on factors like [173]:

- **The number of objectives:** A problem is single-objective if $M = 1$ and multi-objective (or multicriteria/multi-attribute) if $M > 1$. Multi-objective optimization is the predominant type in real-world scenarios.

- **Problem Classification by Function Linearity:** The objective function may be linear or nonlinear. A problem is linearly constrained if its constraints Φ and Ψ are linear. When every function (objective and constraints) defined on the design vector is nonlinear, it's classified as a nonlinear optimization problem.

- **Objective Function Landscape:** The shape (landscape) of a single nonlinear objective function can vary greatly. If it possesses a single valley or peak (a unique global optimum), the problem is unimodal. Here, the single local optimum is the global optimum. However, most functions are multimodal (having multiple optima), making them harder to optimize.

- **Presence of constraints:** Optimization problems are classified as constrained or unconstrained based on the existence of constraints. An unconstrained problem has no constraints whatsoever, meaning $J = 0$ and $K = 0$ (i.e., zero inequality and equality constraints). A problem is constrained when $J + K > 0$ (at least one constraint exists).

- **Design Variable Domains:** discrete optimization occurs when all variables take discrete values (or are reducible to discrete values), while continuous optimization involves variables

that are continuous, taking real values within intervals. Discrete optimization is further divided into two categories: integer programming, where variables are restricted to integer values, and combinatorial optimization, which entails searching for an optimal object within a finite collection of objects possessing a concise representation, such as a graph.

- **Data uncertainty:** The preceding discussions assume deterministic optimization problems, where design variables, objective functions, and constraints possess exact values without uncertainty or noise. In such cases, evaluating any set of design variables yields precise, repeatable results for objectives and constraints. Conversely, if uncertainty or noise affects design variables, objectives, and/or constraints, the problem transitions to stochastic optimization or robust optimizationframeworks specifically designed to handle probabilistic or uncertain elements.

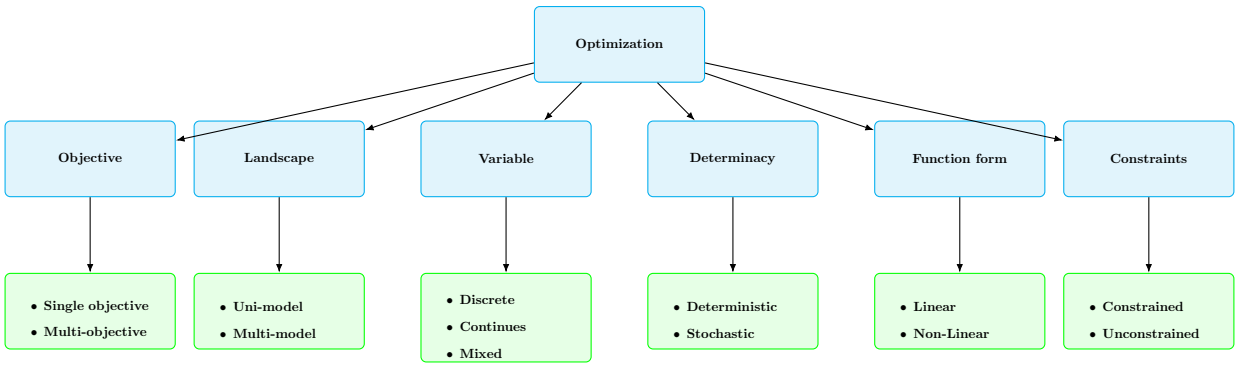


Figure 2.8: Classification of optimization problem.

2.5 Computational complexity

Complexity theory mathematically evaluates optimization problem difficulty, asking why some problems are computationally hard and others easy. As a branch of computation theory, it focuses on minimal resource requirementsmeasured through time complexity (counting computational steps needed; higher steps imply greater difficulty) and space complexity (measuring memory resources required)[142][146].

Complexity theory primarily aims to address these key objectives [7]:

- Establish formal definitions for what constitutes "efficient" problem-solving.
- Classify computational problems as either efficiently solvable or intractable.

- Quantify the resources (time or memory) required to solve problems.

Algorithmic efficiency fundamentally concerns an algorithm's consumption of computational resources, most commonly measured by its running time. This runtime inherently depends on both the size and characteristics of the input. However, certain techniquessuch as randomized algorithmsexhibit variable performance even for identical inputs or across inputs of identical size. Unlike deterministic methods, randomized algorithms incorporate deliberate randomness during execution, meaning their behavior (and thus efficiency) can vary across repeated runs with the same input.

The next section reviews essential complexity theory concepts used throughout this thesis to analyze problem-solving methods.

2.5.1 Definitions and basic concepts

Definition 1 (Algorithm)**.** *[85] An algorithm is a precise sequence of instructions that solves a problem. Given any valid input, it produces the required output within a finite time by executing a unique, finite series of basic operations.*

Definition 2 (Big-O Notation)**.** *[39] For functions $f, g : D \to \mathbb{R}^+$ where n is the independent variable, f is $\mathcal{O}(g)$ if there exist positive constants c and n_0 such that:*

$$0 \le f(n) \le c \cdot g(n) \quad \forall\, n \ge n_0.$$

This indicates $f(n)$ is asymptotically upper-bounded by a constant multiple of $g(n)$.

Geometric Interpretation: For $n \ge n_0$, $f(n)$ lies on or below $c \cdot g(n)$ (illustrated in Figure 2.9). Big-O notation concisely expresses algorithmic running times by capturing **worst-case complexity**the maximum execution time for any input of size n. This simplification is essential since exact running times often involve complex expressions and are typically estimates.

Example: Consider $f(n) = 2n^2 + 5n + 3$. We demonstrate $f(n) = \mathcal{O}(n^2)$: For all $n \ge 1$,

$$f(n) = 2n^2 + 5n + 3 \le 2n^2 + 5n^2 + 3n^2 = 10n^2.$$

Choosing $c = 10$ and $n_0 = 1$ satisfies $f(n) \le c \cdot n^2$ for $n \ge n_0$.

Definition 3 (Time Complexity of an Algorithm)**.** *Consider an algorithm A that processes inputs from a set X, and let $f : \mathbb{N} \to \mathbb{R}^+$ be a function. We say A runs in $\mathcal{O}(f)$ time (or has $\mathcal{O}(f)$ time complexity) if there exist positive constants c and n_0 such that for every input $x \in X$ of size $n \ge n_0$, the **worst-case running time** of A is bounded by $c \cdot f(n)$ elementary operations.*

Definition 4 (Polynomial Time)**.** *An algorithm runs in **polynomial time** if its time complexity is $\mathcal{O}(n^k)$ for some fixed $k \in \mathbb{Z}^+$, where n denotes input size. This class includes:*

- *Functions like $\mathcal{O}(n \log n)$ (since $n \log n \in \mathcal{O}(n^2)$)*

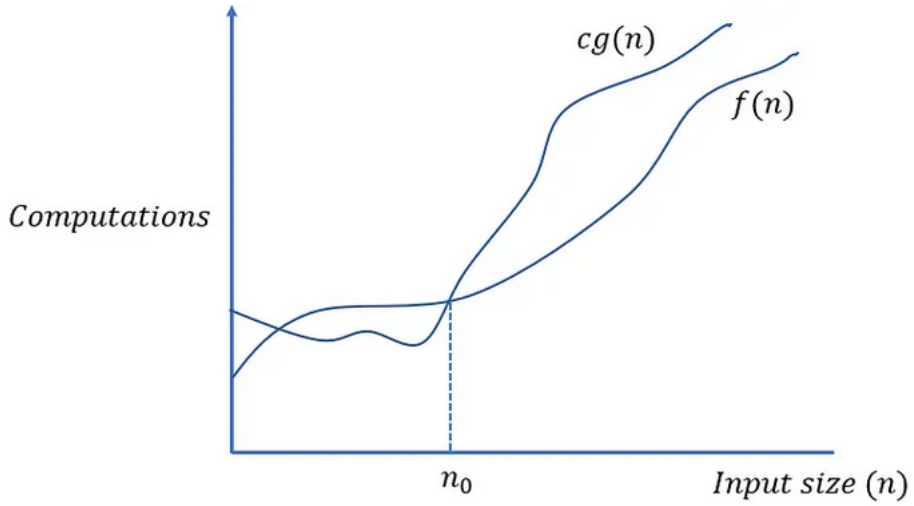


Figure 2.9: Visual illustration of Big-O notation

- *Faster cases: linear ($\mathcal{O}(n)$), logarithmic ($\mathcal{O}(\log n)$), and constant time ($\mathcal{O}(1)$)*

A computational problem is:

- ***efficiently solvable*** *(or* ***tractable****) if solvable in polynomial time*
- ***intractable*** *otherwise*

Definition 5 (Exponential Time)**.** *An algorithm runs in* ***exponential time*** *if its time complexity is $\mathcal{O}(k^n)$ for some fixed $k \in \mathbb{Z}^+$, where n denotes input size. This complexity class exhibits exponential growth: when n increases linearly, the running time grows exponentially.*

Definition 6 (Problem Instance)**.** *[39] For a problem P, an* ***instance*** *I is a specific input configuration that:*

1. *Satisfies all constraints specified in P's definition*
2. *Contains concrete values assigned to all parameters of P*
3. *Provides sufficient information to compute a solution to P*

Definition 7 (Performance Guarantee)**.** *[85] For an optimization problem P with non-negative weights and $k \geq 1$, a* ***k-factor approximation algorithm*** *is a polynomial-time algorithm A satisfying:*

1. ***Maximization:*** $\frac{1}{k} \cdot OPT(I) \leq A(I) \leq OPT(I)$
2. ***Minimization:*** $OPT(I) \leq A(I) \leq k \cdot OPT(I)$

for all instances I of P. The value k is called the ***performance ratio*** *(or* ***performance guarantee****). Special cases:*

- *When* $OPT(I) = 0$*, A must return an exact solution*
- 1*-factor approximations are exact polynomial-time algorithms*

2.5.2 Complexity classes

Computational Complexity Theory classifies problems by their inherent difficulty. As defined previously, a problem's computational complexity quantifies the minimal resources (typically time or space) required for its solution, determined by the most efficient known algorithm. Complexity classes group problems solvable under similar resource constraints [7, 78].

As illustrated in Figure 2.10, four fundamental complexity classes are:

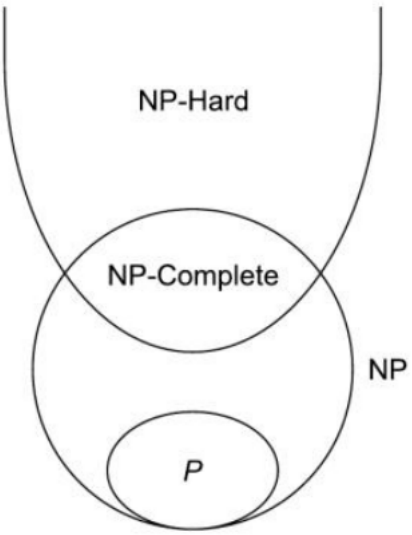


Figure 2.10: the relationship among the complexity classes of problems(NP ≠ P)

- P: Problems solvable in polynomial time
- NP: Problems verifiable in polynomial time
- NP**-hard**: Problems at least as hard as all NP problems
- NP**-complete**: NP-hard problems contained in NP

Recall that a **decision problem** has exactly two possible outputs: *yes* or *no*. For example, determining whether a graph contains a **Hamilton cycle**a simple cycle visiting each vertex exactly once.

2.5.2.1 Class P

The complexity class P comprises all decision problems solvable in polynomial time. Formally, a decision problem belongs to P if there exists an algorithm that solves any instance of the problem in $O(n^k)$ time for some constant $k \in \mathbb{N}$, where n denotes the input size.

2.5.2.2 Class NP

The class NP (non-deterministic polynomial time) consists of decision problems where a proposed "yes" answer can be checked for correctness in polynomial time. While a solution can be verified quickly, finding one might be difficult. The class P, containing problems that can be both solved and verified quickly, is a subset of NP. A major unsolved problem in computer science is whether P equals NP; however, the prevailing belief is that they are not equal.

2.5.2.3 Class NP-complete

Assuming $P \neq NP$, the NP-**complete** class comprises the hardest problems in $NP \setminus P$. Formally, a problem is NP-complete if:

1. It belongs to NP
2. Every problem in NP is polynomial-time reducible to it (NP-hard)

This class has a fundamental property: if *any* NP-complete problem admits a polynomial-time solution, then P = NP. Despite extensive research, no polynomial-time algorithm has been discovered for any NP-complete problem [39].

2.5.2.4 Class NP-hard

The class NP-**hard** contains problems satisfying:

1. Not necessarily in NP
2. All NP problems are polynomial-time reducible to them

These problems are *at least as hard* as any in NP. This class generalizes NP-complete problems, including optimization/search variants. Crucially, NP-hardness does not imply all instances are equally difficult. Practical approaches include [70]:

1. **Special-case optimization**: Identify application-relevant subclasses solvable in polynomial time
2. **Approximation algorithms**: For cases without exact polynomial solutions, use algorithms with:

- Polynomial runtime
- Provable performance guarantees (approximation ratios)

3. **Stochastic methods**: When approximation is infeasible, employ randomized algorithms

These approaches constitute the primary methodologies for tackling NP-hard problems, detailed in subsequent sections.

2.6 Local Versus Global Optima

Optimization solutions are characterized by solution quality, primarily classified as **local optima** or **global optima**. Assuming minimization without loss of generality, consider optimizing an objective function $f(x)$ over feasible solutions $x \in \mathscr{F} \subseteq S$.

Global Minimum: A solution $x^* \in \mathscr{F}$ is a global minimum if:

$$f(x^*) \leq f(x) \quad \forall\, x \in \mathscr{F}$$

This represents the absolute best solution in the feasible set. Note that multiple global minima may exist.

Local Minimum: A solution $x_N^* \in \mathscr{F}$ is a local minimum if $\exists \epsilon > 0$ such that:

$$f(x_N^*) \leq f(x) \quad \forall\, x \in \mathscr{F} \cap \mathscr{N}_\epsilon(x_N^*)$$

where $\mathscr{N}_\epsilon(x_N^*) = \{x \in S : \|x - x_N^*\| \leq \epsilon\}$ is an ϵ-neighborhood. This indicates optimality within a local region.

Global minima represent the overall best solutions, while local minima are optimal only within their immediate neighborhoods.

The intersection $\mathscr{F} \cap \mathscr{N}$ represents the feasible solutions within the ϵ-neighborhood of x_N^*. Visual examples of local and global optima are provided in Figure 2.11 [12].

2.7 Optimization methods

This thesis focuses on combinatorial optimization problems (COPs), a highly active research area. While conceptually simple to model, COPs are notoriously difficult to solve because the number of feasible solutions grows exponentially with problem size. Consider the classic Knapsack Problem: selecting items (each a binary choice) for a knapsack of limited capacity. A problem with N items has 2^N possible solutions. Exhaustive search, evaluating every solution to find the best, is feasible for small N (e.g., $N = 10$ has $1,024$ solutions). However, for larger instances (e.g.,

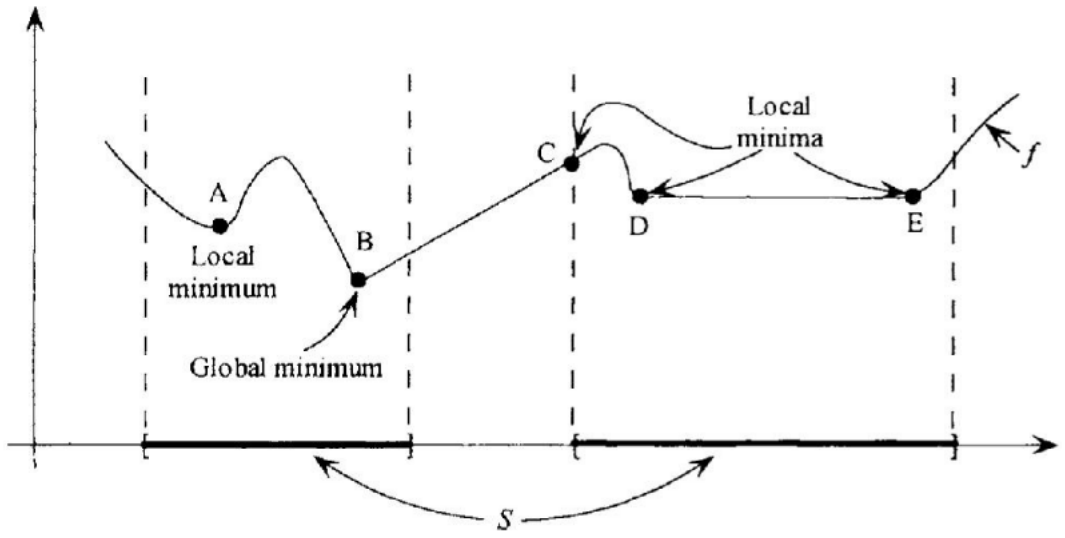


Figure 2.11: Local Versus Global Optima

$N = 100$, yielding approximately $1.26e30$ solutions), exhaustive search becomes computationally prohibitive, limiting its practicality to small problems.

Solving combinatorial optimization problems typically involves two distinct strategies [50]:

- **Exact Algorithms:** Guarantee finding the optimal solution.
- **Approximate Algorithms:** Used for larger or more complex instances where exact methods are impractical; these find near-optimal solutions.

Approximate algorithms are subdivided into heuristic and metaheuristic methods. Notably, when an approximate algorithm has a provable performance guarantee (i.e., its solutions are never worse than the optimum by more than a fixed amount or factor), it is classified as an approximation algorithm[46].

2.7.1 Exact Algorithms

Exact methods (also called complete algorithms) guarantee finding an optimal solution and verifying its optimality for any finite-sized combinatorial optimization problem instance within bounded time. However, most typical combinatorial problems are difficult (often lacking polynomial-time algorithms), and the computation time for exact methods typically grows exponentially with problem size. This renders them impractical for large-scale real-world instances, limiting their suitability to smaller problems. Well-known exact methods include the Branch-and-X family (Branch and Bound, Branch and Cut, Branch and Price), Dynamic Programming, Linear Programming, and specialized algorithms developed for specific problems.

2.7.2 Approximate Algorithms

Approximate algorithms aim to deliver solutions close to the optimal for computationally challenging problems, particularly when exact methods are prohibitively slow or impractical. These algorithms are extensively applied in areas such as combinatorial optimization, scientific computing, and machine learning, where calculating the precise solution becomes impossible for large-scale problem instances[155].

For an optimization problem, an approximate algorithm guarantees that its solution's quality is always within a defined factor (known as the approximation ratio) of the true optimum value. Specifically:

- For maximization problems, the solution value is at least α times the optimal value (where $\alpha \leq 1$).
- For minimization problems, the solution value is at most β times the optimal value (where $\beta \geq 1$).

As categorized by Talbi[50], approximate algorithms fall into two main groups: specific heuristics and metaheuristics.

2.7.2.1 Heuristics

Originating from the ancient Greek verb heuriskein (meaning "to find"), a heuristic is a problem-solving approach designed to deliver good, near-optimal solutions efficiently. It operates within reasonable computational limits but cannot guarantee that solutions will be feasible or optimal. Often, it cannot even determine how close a feasible solution is to the true optimum. Yang [173] further defines a heuristic as a trial-and-error strategy that yields acceptable solutions for complex problems within practical timeframes. Given the problem's complexity, which makes exhaustive search impossible, heuristics aim to find high-quality, feasible solutions quickly. While there's no assurance of finding the best solution or even fully understanding why a specific heuristic works, these methods are typically tailored to specific problems. Consequently, a heuristic effective for one problem may not necessarily solve another. The core objective is an efficient, practical algorithm that consistently produces good solutions. Among these, some may be nearly optimal, although optimality itself is never guaranteed.

Why use heuristics?

The primary motivation is the difficulty or impossibility of finding optimal solutions for many common models. However, several other compelling reasons exist [155]:

- Easier Implementation: Simpler to put into practice.
- Performance Improvement: Can demonstrably outperform existing methods.
- Speed: Delivers results rapidly.

- Robustness: Tends to perform reliably under varying conditions.
- Integration: Useful components within larger optimization procedures.

Classification of heuristic:

Heuristic methods can be categorized in several ways. Below are some common types with brief definitions [155]:

- Random Solution Generation: A simple approach where feasible solutions are generated randomly, evaluated, and the best-performing one is selected.
- Problem Decomposition/Partitioning: This method breaks down a complex problem into smaller, more manageable subproblems presumed easier to solve. Partitioning can occur based on decision hierarchies (e.g., design vs. operation), key resources (e.g., different machines in production scheduling), or the chronological sequence of decisions.
- Inductive Methods: These involve generalizing solutions from either smaller/simpler versions of the same problem or mathematically analogous problems. The latter utilizes analogy, a key element in creative problem-solving.
- Solution Space Reduction Methods: The core idea is to drastically limit the number of solutions considered, aiming to preserve solution quality. One strategy is to quickly eliminate solutions (i.e., variable value combinations) that violate the problem's constraints.
- Approximation Methods: These methods specifically manipulate the mathematical model itself or use solutions from related, simpler models. (Note: The original text mentions four categories here but doesn't elaborate).
- Constructive Methods: As the name suggests, these build a solution incrementally using the problem's data. Typically, no complete solution exists until the procedure finishes (unlike improvement methods). A specific type is the greedy method, where at each step, the next solution element is chosen to maximize immediate benefit (e.g., highest profit or lowest cost). This myopic, step-by-step approach resembles sequential decision-making in decomposition methods.
- Local Improvement (Neighborhood Search) Methods: This concept starts with a feasible solution (often from a constructive method). It then evaluates neighboring solutions around the current solution. If a better solution is found in this neighborhood, it replaces the current solution, and the search continues from this new point.

2.7.2.2 Metaheuristics

Metaheuristics are high-level problem-solving frameworks designed for diverse optimization challenges. Coined by Glover [61] from the Greek meta ("beyond") and heuristic, they orchestrate lower-level heuristics to enhance performance. Key definitions characterize them as[17]:

- Iterative processes that intelligently guide subordinate heuristics by balancing search space exploration and exploitation, using learning strategies to efficiently find near-optimal solutions.
- Strategic controllers that help underlying heuristics escape local optima either by permitting temporary worsening moves or generating intelligent restart points thereby overcoming limitations of basic iterative methods. They introduce biases (e.g., descent, memory, or experience-based) to accelerate discovery of high-quality solutions. Crucially, while they incorporate probabilistic decisions, their use of randomness is purposefully directed not blind, as in random search.

Most leading metaheuristic algorithms were developed before 2000, termed "classical" [45]. Key examples include Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization, Genetic Programming (GP), Differential Evolution (DE), Simulated Annealing (SA), Tabu Search (TS), GRASP, Artificial Immune Algorithm (AIA), Iterated Local Search (ILS), Chaos Optimization (COM), Scatter Search (SS), Shuffled Frog-Leaping (SFLA), and Variable Neighborhood Search (VNS).

Despite their success, the last two decades have seen a surge of novel evolutionary approaches. These new metaheuristicsinspired by natural processesoften outperform classical methods on unsolved benchmarks(ref). Notable examples include Artificial Bee Colony (ABC), Bacterial Foraging (BFO), Bat Algorithm (BA), Biogeography-Based Optimization (BBO), Cuckoo Search (CS), Firefly Algorithm (FA), Gravitational Search (GSA), Grey Wolf Optimizer (GWO), Harmony Search (HS), Social Spider Optimization (SSO), and Whale Optimization Algorithm (WOA).

2.7.2.3 Classification of metaheuristics

Metaheuristics are categorized in academic literature according to various criteria. The most frequent methodclassifying by source of inspiration, such as natural or physical processesoffers limited insight into the mathematical principles governing the algorithms. A more informative approach categorizes them based on the number of search agents, distinguishing between trajectory-based and population-based methods, which indicates how many candidate solutions are updated each iteration. Alternatively, Molina et al. [111] suggest grouping algorithms by their behavioral mechanisms, such as (a) Differential Vector Movement or (b) Solution Creation, which provides a clearer view of their internal functioning. Another taxonomy is proposed in [131]. Given that algorithmic parameters greatly influence performance and manual tuning

remains challenging due to a lack of universal rules, this latter study offers a novel classification system based on the number of parameters required [131].

In [17], Blum provides a concise overview of the primary classification schemes for metaheuristics:

- Metaheuristics are often divided into nature-inspired (e.g., Genetic Algorithms, Ant Colony Optimization) and non-nature-inspired (e.g., Tabu Search, Iterated Local Search) categories. However, this classification offers limited practical utility for two primary reasons. First, contemporary hybrid algorithms frequently blend elements from both categories, defying clear classification. Second, the distinction itself can be ambiguous; for instance, one could argue that the memory function in Tabu Search is metaphorically inspired by human cognition, a natural process. These ambiguities ultimately diminish the framework's usefulness.

- Population-based (multiple solutions) vs. single-point search (one solution): Single-point algorithms, termed trajectory methods (e.g., Tabu Search, Iterated Local Search, Variable Neighborhood Search), trace a path through the search space. Conversely, population-based metaheuristics model the collective evolution of a solution set within the search space.

- Static vs dynamic: Static methods use the original function unchanged, while dynamic methods (like Guided Local Search) actively modify it during the search. This strategy aims to escape local minima by altering the search landscape, incorporating information gathered throughout the process.

- Single-neighborhood vs. Multi-neighborhood: Most algorithms operate using a single, fixed neighborhood structure, resulting in an unchanging fitness landscape topology. Others, like Variable Neighborhood Search (VNS), utilize multiple predefined neighborhood structures. This allows them to actively switch between different landscape topologies during the search, enhancing diversification.

- Metaheuristics can be classified by their use of memory (search history). Memory usage is now considered a fundamental feature of effective metaheuristics. Memory-less algorithms operate as a Markov process, where only the current search state determines the next move. Memory-based algorithms, conversely, utilize stored information, typically categorized as:

  1. Short-term memory: Tracks recent actions (e.g., moves, visited solutions).
  2. Long-term memory: Stores aggregated statistics or synthesized search data.

2.8 Conclusion

This chapter offers a comprehensive survey of various network models, encompassing simple, weighted, directed, signed, multiplex, and dynamic networks. Subsequently, it delivers a thorough

examination of optimization problems, beginning with fundamental definitions of combinatorial optimization and a classification based on key attributes. The chapter concludes with a review of heuristics and meta-heuristics, including their taxonomy, with specific focus on the techniques relevant to our research.

CHAPTER 3

EVOLUTION OF OPTIMIZATION-BASED COMMUNITY DETECTION

"Social network analysis focuses on relationships among social entities and the patterns and implications of these relationships".Wasserman

3.1 Introduction

Community detection in complex networks is a critically important yet NP-hard problem, with significant applications in social, biological, and network design fields. This chapter first defines the problem across various network models (simple, multiplex, dynamic) and reviews key heuristic and metaheuristic detection algorithms. We then propose a new taxonomy focused on hybrid and incremental strategies to guide future research. Most importantly, we introduce a systematic framework that integrates these algorithms with network embedding techniques, outlining promising new research directions to develop more powerful solutions.

3.2 Problem formulation

3.2.1 Problem Statement: Community Discovery Problem

The community discovery task is formally defined as:
The community detection objective partitions vertex set V into k non-overlapping communities in graph G, yielding $C = \{C_1, C_2, \ldots, C_k\}$ where:

- $\bigcup_{i=1}^{k} C_i = V$ (vertex coverage)

- $C_i \cap C_j = \emptyset \ \forall \ i \neq j$ (pairwise disjointness)

Formally defining edge relationships:

- Intra-community edges for C_i: $\mathscr{E}_{c_i}^{in} = \{(u,v) \in E \mid u,v \in C_i\}$, can be formally define as follow:

$$\mathscr{E}_{C_i}^{in} = \frac{\sum_{i \in C_k} \sum_{j \in C_k} A_{ij}}{2} \tag{3.1}$$

- Inter-community edges for C_i: $\mathscr{E}_{c_i}^{out} = \{(u,v) \in E \mid u \in C_i, v \notin C_i\}$, can be formally define as follow:

$$\mathscr{E}_{C_i}^{out} = \sum_{i \in C_k} \sum_{j \notin C_k} A_{ij} \tag{3.2}$$

The total degree $\mathscr{D}_{C_i}$ of community C_i is given by:

$$\mathscr{D}_{C_i} = \sum_{i \in C} \sum_{j=1}^{n} A_{ij} \tag{3.3}$$

3.2.2 Problem Statement: Overlapping Community Discovery

The community detection objective partitions vertex set V into k overlapping communities in graph G, yielding $C = \{C_1, C_2, \ldots, C_k\}$ where:

- $\bigcup_{i=1}^{k} C_i = V$ (vertex coverage)
- The nodes of V may belong to multiple communities, $C_i \cap C_j \neq \emptyset$[89].

3.2.3 Problem Statement: Multiplex Community Discovery

Loe and Jensen [101] expanded on Fortunato's [54] definition of community structure to include multiplex networks. Nonetheless, multiplex community detection needs a broadly accepted formal definition, as its application frequently varies according to the specific issue field. Their framework categorizes community identification methodologies under three paradigms: local, global, and node similarity-driven criteria.

- Local definitions emphasize the transmission of information among nodes, suggesting redundancy as a metric for evaluating community similarity throughout the network's layers.
- Global criteria evaluate emergent community structures via network-scale measures including modularity.
- Definitions based on node similarity highlight the resemblance of nodes within communities across layers, utilizing criterion like the cross-layer edge clustering coefficient.

Definition 1: Consider $G_M = \{G_l; l \in 1,...,d\}$. There exists a k subset partitions $C_k \subseteq V_M$ of node-layer pairs of a multiplex network such that $C_1 \cup ... \cup C_k = V_M$. Let $C_k^l = C_k \cap V_M$ denote the set of node-layer pair of C_k present in layer $l = 1,.....,L_d$ [18].

3.2.4 Problem Statement: Dynamic Community Discovery

As mentioned above there are different types of networks model. Thus, we formally define the dynamic community detection problem following these models:

Definition 1(Community Stream): $\mathbb{C} = \{C_1,\ldots,C_t\}$, where C_t comprises all communities detected in network snapshot G_t.

Definition 2: Let C_t represent the set of community structures detected of network G_t at timestep t. That is $C_t = \{C_t^1, C_t^2, ..., ..C_t^k\}$ where :

- K corresponds to the cardinality of the community set within G_t.
- C_t^i corresponds to community i within snapshot G_t.
- $C_t^i \cap C_t^l$ is not necessarily empty.

Definition 3: Akin to the temporal network model. Consider the graph $G_t = (T,V,W,E)$, the detected community C_{ith}^t of G_t is defined as follow C_i^t is a subset of W. $E(C)$ is defined as the set of links between nodes involved in E where $E(C) = \{(t,uv) \in E, (t,u) \in C, (t,v) \in C\}$.

Definition 4: For tracking community evolution and topological transformations over time, events must be defined to characterize the dynamic behavior of social networks. Several event types are introduced here 3.2.4.1.

3.2.4.1 Community dynamic life cycle

Local and atomic transformations involved in dynamic networks, such as node creation and deletion or edge rewiring, can result in a set of events characterized by these changes. Several works in the literature attempt to define such events[124] [29]. Palla et al [124]. first introduced seven core events: Birth, Death, Growth, Contraction, Merge, Split, and Continue. Later, a study in [34] Cazebet et al expanded this framework by adding an event termed Resurgence.

These events, exhibited in Fig3.1, are following bellow:

- Growth: The process whereby communities increase in scale through node addition.
- Contraction: Community size reduction resulting from node removal.
- Merging: The combination of two or more communities into a single entity.

- Split: Partitioning of a community into multiple discrete subgroups.
- Birth: The emergence of a new community at a specific time, comprising any number of nodes.
- Death: The dissolution of a community at a specific time, marked by the loss of all its constituent nodes
- Resurgence: The temporary dissolution of a community followed by its seamless reformation after a time-delayed period, retaining the same set of nodes. This event represents a combined death-birth sequence, where the community appears to disband but later reconstitutes identically (e.g., seasonal behaviors or periodic activity).

3.2.5 The instability of detected community partitions in dynamic networks and the temporal smoothness:

A critical limitation in dynamic community detection is partition instability: no unique community decomposition exists, and algorithms may yield divergent partitions for near-identical network states. This occurs both when networks undergo minor topological changes andin stochastic algorithmseven without structural variations. Consequently, partition selection becomes fundamentally arbitrary. Modularity-optimization approaches exemplify this issue, where multiple high-scoring partitions may exist for a single snapshot [138].

These challenges underscore the fundamental problem of identifying stable network decompositions in dynamic community detection. A central difficulty lies in disambiguating true evolutionary changes (communities evolving from t to $t+1$) from algorithmic instability artifacts. To mitigate such instabilitymanifested as clustering driftsresearchers [33] have developed smoothness-based techniques that enforce evolutionary continuity. These methods are taxonomically divided into global and local smoothness approaches.

1. **The global smoothness technique:** For each time step communities are defined by smoothing all the snapshots of networks. One such strategy involves optimizing a global metric across all snapshots simultaneously. Another approach is to construct a unified network by aggregating snapshotsfor example, by adding inter-snapshot links between corresponding nodesand then applying a community detection algorithm to this integrated structure. One of the advantages of global smoothness is its strong computational performance; however, this approach is constrained by its time complexity.
2. **The local smoothness techniques**: Operating via second-order smoothness, these methods measure clustering drift between adjacent snapshots alone, optimizing the performance-complexity tradeoff.

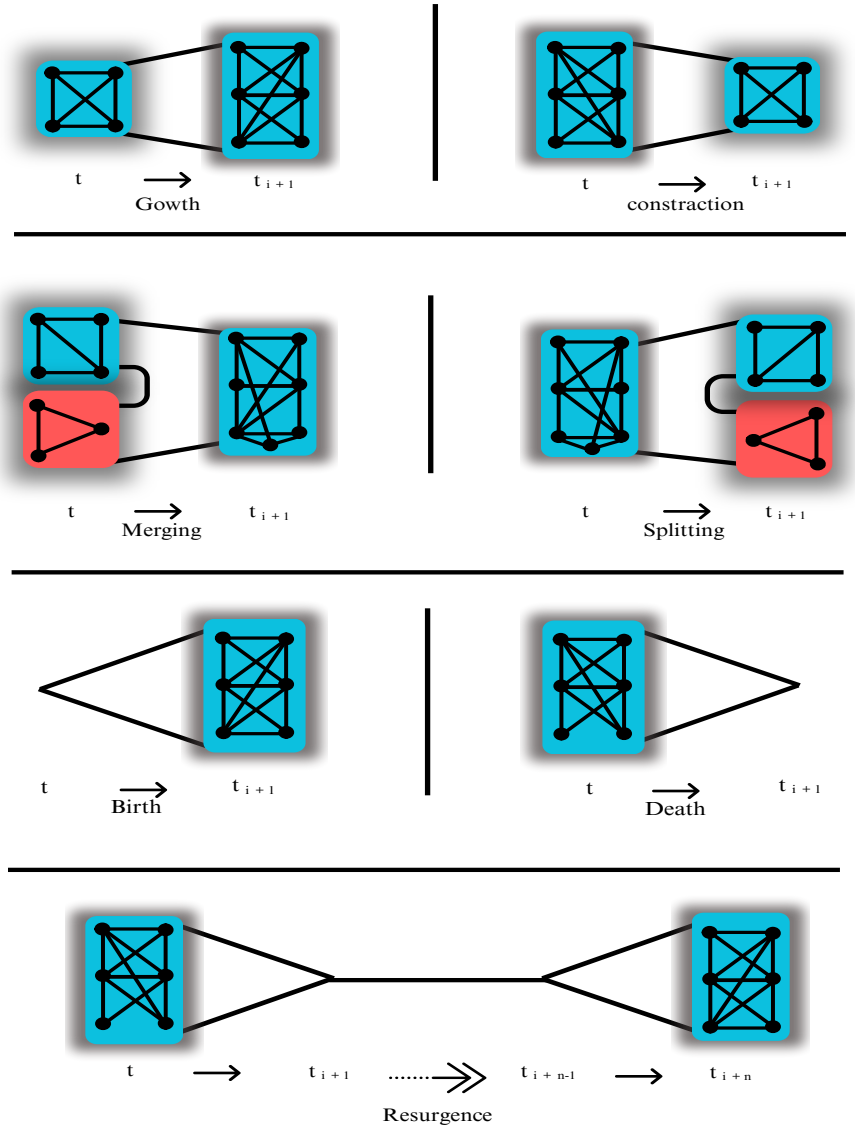


Figure 3.1: **Fundamental Community Dynamics:** This illustrative model captures eight core events governing community evolution in dynamic networks: (1) Birth and Death (emergence/dissolution of communities); (2) Growth and Contraction (expansion/shrinkage of membership); (3) Merge and Split (fusion/fragmentation of communities); (4) Continue (structural persistence across time); and (5) Resurgence (reemergence of historically dissolved communities)[34].

a) Implicit Smoothing: Certain approaches inherently promote partition similarity across consecutive timesteps without explicit constraints. While not directly incorporating inter-snapshot similarity metrics, their design intrinsically favors temporal continuity. Two distinct categories emerge:

   i. Methods maximizing global metrics (e.g., modularity) initialize the current iteration's community detection using prior-iteration communities as seed configurations, accelerating convergence while preserving evolutionary continuity.

   ii. Methods iteratively adapt community structures per timestep through predefined local rules triggered by neighborhood-level topological changes (e.g., edge additions/removals).

b) Explicit Smoothing: Methods in this category impose direct similarity constraints between consecutive partitions. Specifically, the solution P_t must satisfy $Sim(P_t, P_{t-1}) \geq \theta$ where θ is a similarity threshold. These approaches incorporate a trade-off parameter $\alpha \in [0,1]$ that explicitly balances current-snapshot optimality against temporal consistency with P_t .

The decision to adopt a specific smoothing strategy critically influences both computational constraints and result quality. To address this, we designed a novel taxonomy for community detection optimization that leverages the role of smoothing to distinguish between different classes of dynamic community detection approaches in the next section.

3.2.6 Community detection as optimization problem:

The identification of community structures is fundamentally an optimization problem, addressable through two paradigms: single-objective or multi-objective formulation[52][129]. Let $\mathscr{C}^* = \{C_1, \ldots, C_r\}$ denote the set of feasible identified communities within a network.

Under single-objective optimization, community detection is modeled as the optimization problem $(\mathscr{C}^*, \mathscr{F})$, seeking an optimal partition $\mathscr{C}^*$ where:

$$\mathscr{F}(C^*) = min\mathscr{F}(C) \quad subject, to C \in \mathscr{C}^* \tag{3.4}$$

Where $\mathscr{F} : \mathscr{C}^* \rightarrow \mathbb{R}$ denotes the single-objective function evaluating partition feasibility and community quality. For multi-criterion optimization, this extends to a multiobjective clustering formulation. $(\mathscr{C}^*, \mathscr{F}_1, \mathscr{F}_2, \ldots, \mathscr{F}_h)$

$$\mathscr{F}(C^*) = min\mathscr{F}_i(C) \quad subject, to C \in \mathscr{C}^*. \tag{3.5}$$

where $\mathscr{F} = \{\mathscr{F}_1, \ldots, \mathscr{F}_r\}$ denotes a collection of competing single-criterion functions $\mathscr{F}_i : \mathscr{C}^* \rightarrow \mathscr{R}$ that require simultaneous optimization. The objective is to identify an efficient solution, as no dominant solution exists that optimally satisfies each criterion. The theory of Pareto optimality

[49] holds the identification of these solutions. For solutions $C1, C2 \in \mathscr{C}^*$, $C1$ dominates $C2$ (denoted $C1 < C2$) if and only if:

$$\forall i : \mathscr{F}_i(C_1) \leq \mathscr{F}_i(C_2) \exists i \, s.t \mathscr{F}_i(C_1) < \mathscr{F}_i(C_2) \tag{3.6}$$

Multi-objective optimization seeks to generate and select non-dominated solutions, termed Pareto-optimal, where enhancement of one objective necessitates a compromise in another. The collection of Pareto-optimal solutions π is defined as:

$$\pi = \{C \in \mathscr{C}^* : \nexists C_0 \in \mathscr{C}^* \, with \, C_0 < C\} \tag{3.7}$$

3.3 Evaluation Metrics:

Two distinct mechanisms have been proposed to assess the quality of detected communities. The first involves applying quality scoring functions to evaluate the connectivity of communities. This mechanism is categorized into two approaches: one that relies on the existence of ground-truth data and another that operates independently of ground truth, using intrinsic structural metrics like modularity or cohesion.

The second mechanism focuses on analyzing the topological properties of identified communities (e.g., size, density, or hierarchy) and assessing the scalability and computational complexity of the detection method itself. Together, these frameworksexternal/internal quality scoring and topological/complexity analysisenable a comprehensive evaluation of the performance of community detection algorithms[138].

3.3.1 External approaches based on the ground truth

Ground-truth communities are predefined node groupings in a network, established using two primary criteria: Structural (based on measurable properties like internal density or modularity) and Semantic (based on external node attributes like shared labels or functions). The performance of community detection algorithms is typically assessed by measuring how closely their output matches this known ground-truth structure. One of the most common used similarity measures is normalized mutual information NMI[42].

$$\mathrm{NMI}(C, C') = \frac{-2 \sum_{i=1}^{M_C} \sum_{j=1}^{M_{C'}} M_{ij} \log\left(\frac{n \cdot M_{ij}}{M_{i.} M_{.j}}\right)}{\sum_{i=1}^{M_C} M_{i.} \log\left(\frac{M_{i.}}{n}\right) + \sum_{j=1}^{M_{C'}} M_{.j} \log\left(\frac{M_{.j}}{n}\right)} \tag{3.8}$$

The (NMI) metric quantifies partition similarity on a [0,1] scale, where values approaching 1 indicate near-perfect alignment between detected communities and ground-truth structures. This measure is essential for evaluating an algorithm's accuracy in recovering known network communities. Let C' represent the actual community structure and C denote the identified divisions. Let $M = [M_{ij}]^{C \times C'}$ be the confusion matrix

where: Rows correspond to ground-truth communities ($i = 1,\ldots,C$) ; columns represent detected communities ($j = 1,\ldots,C'$); element M_{ij} counts nodes from true community i assigned to detected community j; C and C' denote the cardinality of ground-truth and detected partitions respectively. To address selection bias, when the division solution contains numerous small partitions or insufficient nodes relative to the ground-truth solution, Romano et al. [136] proposed the weighted normalized mutual information NMI_w.

$$\mathsf{NMI}_w(C,C') = e^{\frac{-|c'-c|}{c'}} \times \mathsf{NMI}(C,C') \tag{3.9}$$

For overlapping structures, Lancichinetti et al. [89] developed NMI_{over}, extending NMI's similarity assessment capabilities.

$$\mathsf{NMI}_{over}(\mathbb{P},\mathbb{P}') = \frac{H(\mathbb{P}) + H(\mathbb{P}') - H(\mathbb{P},\mathbb{P}')}{(H(\mathbb{P}) + H(\mathbb{P}'))/2} \tag{3.10}$$

where $H(\mathbb{P})$ is the entropy of the random variable associated with the detected community partition, $H(\mathbb{P}')$ is the entropy of the random variable corresponding to the ground truth partition, and $H(\mathbb{P},\mathbb{P}')$ denotes their joint entropy.

Community stability was evaluated via NMI-based temporal coherence analysis across network snapshots[33].

$$\mathsf{NMI}_m = 1 - \frac{1}{T-1}\sum_{t}^{t-1} \mathsf{NMI}(G_t, G_{t+1}) \tag{3.11}$$

A key limitation of Normalized Mutual Information (NMI) is its quadratic complexity $O(z^2)$ for large communities, prompting development of scalable alternatives. In this regard, Rossetti et al [140] introduced the F1-community score, a metric that aims to reduce computational burden while retaining evaluations rigor.

$$F1 = 2 \times \frac{Precision * Reacll}{Precision + Reacll} \tag{3.12}$$

An evaluation score is generated through macro-averaged F1-measurement: first computing precision and recall per community, then taking the harmonic mean for each, and finally averaging across the community set.

Community Precision: the proportion of vertices in algorithm community x that are classified with ground truth community y, calculated as:

$$Precsion = \frac{|X \cap Y|}{|X|} \tag{3.13}$$

Community Recall: the proportion of vertices within algorithm community x that are classified as belonging to ground truth community y, calculated as:

$$Recall = \frac{|X \cap Y|}{|Y|} \tag{3.14}$$

This evaluation approach faces substantive criticisms [127]. The planted partition model cannot guarantee optimality across all quality functions optimized by detection algorithms, given community discovery's ill-posed nature. Despite these limitations, it remains the predominant benchmarking method for algorithmic comparison.

3.3.2 External approaches based on the quality function:

This approach is recommended when ground-truth communities are unavailable. By employing a community quality score to identify partitionsprioritizing those with the highest scoresthe most effective results can be achieved for the analyzed networks.

3.3.2.1 Score function based on a network model

Modularity [116] remains a cornerstone metric for evaluating community structure in complex networks. It operates by comparing a network's actual structure to a degree-preserving random null model.

The foundation of this model is the expected number of edges between nodes i and j, given by $\frac{d_i d_j}{2m}$ for a network with m edges. This formulation derives from the configuration model:

- The probability that any edge stub from node i connects to node j is $\frac{d_j}{2m}$.
- With d_i stubs, the total expected connections from i to j is $\frac{d_i d_j}{2m}$.

The term $A_{ij} - \frac{d_i d_j}{2m}$ thus quantifies the deviation of the observed connectivity (the adjacency matrix element A_{ij}) from this random baseline.

Formally, modularity for a partition C is defined as:

$$Q(C) = \frac{1}{2m} \sum_i \sum_j \left(A_{ij} - \lambda \frac{d_i d_j}{2m} \right) \delta(C_i, C_j) \tag{3.15}$$

Here, C is the set of all communities. The Kronecker delta function $\delta(C_i, C_j)$ [133] equals 1 if vertices i and j belong to the same community ($C_i = C_j$) and 0 otherwise. The resolution parameter λ [134] is introduced to address modularity's well-known resolution limit, which hinders the detection of small communities in large-scale networks.

In essence, Q measures the difference between the actual density of edges within communities and the expected density under the random null model. Higher values indicate stronger, more distinct community structures, with the score normalized to a

range between -1 and 1. It is important to note that modularity maximization is NP-hard [27], even when restricted to specific graph cuts.

In addition, the literature offers alternative versions of Q, along with further formulations tailored to address the varying structures and properties of complex networks. For example, Modularity Q_d[91] for directed networks, Q_w [5] for weighted networks, Q_s[63] for signed networks, $Q_{Multiplex}$[112] for multiplex networks, $Q_{overlapping}$ for overlapping communities [149].

The directed modularity metric has the following formal definition:

$$Q_d(C) = \frac{1}{m}\sum_i\sum_j\left(A^d_{ij} - \frac{d^{in}_i d^{out}_j}{2m}\right)\delta(C_i, C_j) \tag{3.16}$$

Modularity for weighted networks is formally defined as:

$$Q_w(C) = \frac{1}{2w}\sum_i\sum_j\left(w_{ij} - \frac{w_i w_j}{2w}\right)\delta(C_i, C_j) \tag{3.17}$$

Modularity for signed networks is formally defined as:

$$Q_s(C) = \frac{1}{2d^+ + 2d^-}\sum_i\sum_j\left[A^-_{ij} - A^+_{ij} - (\lambda^+\frac{d^+_i d^+_j}{2d^+} - \lambda^-\frac{d^-_i d^-_j}{2d^-})\right]\delta(C_i, C_j) \tag{3.18}$$

Modularity generalization for multiplex networks incorporates layer-specific representations: each layer l is encoded by adjacency matrix A^l_{ij}. The formulation includes:

- λ^l: Layer-specific resolution parameter.
- $\delta(c^l_i, c^l_j)$: Kronecker delta indicating intra-layer community co-membership (1 if nodes i, j share community in layer l, else 0).
- w^{kl}_j: Inter-layer coupling weight connecting node j's replicas across layers k and l.

The coupling parameter w^{kl}_j is typically binary-valued $(0, \omega)$, with $\omega > 0$ if $k \neq l$ and $i = j$ (enforcing same-node inter-layer connections only) [112].

$$Q_{Multiplex}(C) = \frac{1}{2\mu}\sum_{i,j=1}^{n}\sum_{k,l=1}^{L}\left[(A^l_{ij} - \lambda^l\frac{d^l_i d^l_j}{2m^l})\delta_{kl} + \mathrm{w}^{kl}_j\delta_{ij}\right]\delta(C(i,k), C(j,l)) \tag{3.19}$$

An alternative modularity extension evaluates overlapping communities. Defining n_{c_i} and n_{c_j} as the community membership counts for vertices i and j respectively, this overlapping modularity formulation adheres to two fundamental properties:

- Reduction to Standard Modularity: Q_{ov} simplifies to the classical Newman-Girvan modularity Q when each vertex belongs to exactly one community.

- Null Case: Q_{ov} when all nodes belong to the same community (no meaningful structure). For further details, see [119], which formalizes overlapping modularity using membership counts also generalized for overlapping communities in directed weighted graph.

$$Q_{ov}(C) = \frac{1}{2m} \sum_{c \in C} \sum_{i,j \in V} \frac{1}{nc_i nc_j} \left(A_{ij} - \frac{d_i d_j}{2m} \right) \tag{3.20}$$

To address the resolution limit of modularity, a variant termed density modularity has been proposed[96].

$$Q_{Density}(C) = \sum_{c_i \in C} \frac{\mathscr{E}_{c_i}^{in} - \mathscr{E}_{c_i}^{out}}{\mid c_i \mid} \tag{3.21}$$

Each summand corresponds to the ratio of the difference between the number of intra-community edges and inter-community edges to the size of the community.

Several alternative modularity metrics have been proposed in the literature. These metrics are adopted either as objective functions in heuristic methods or as multi-objective criteria in metaheuristics [152]. They assessed the extent to which the connectivity structure of a given set of nodes aligns with that of a community. Based on their criteria, these structures can be categorized into three types: score function-based internal connectivity, score function-based external connectivity, and a combined score function that incorporates both internal and external connectivity[172]. Some well-known examples include:

3.3.2.2 Score function-based internal connectivity

Internal density: Quantify intra-community density, where higher density implies tighter, more coherent divisions. Where k is the number of communities [130].

$$ID(C) = \sum_{i=1}^{k} \frac{\mathscr{E}_{c_k}^{in}}{\mid c_k \mid (\mid c_k \mid -1) \div 2} \tag{3.22}$$

Ratio association: Aims to optimize ratio association for partitioning graphs into small, disjoint, densely connected clusters[4].

$$RA(C) = \sum_{i=1}^{k} \frac{\mathscr{E}_{c_k}^{in}}{\mid c_k \mid} \tag{3.23}$$

To reformulate the maximization of ratio association as a minimization optimization problem, Maoguo Gong et al. proposed **the Negative Ratio Association** (NRA) metric

[64].

$$NRA(C) = -\sum_{i=1}^{k} \frac{\mathscr{E}_{c_k}^{in}}{\mid c_k \mid} \tag{3.24}$$

Kernel k-mean: The minimization objective of kernel k-means[44][4] can be directly applied to identify the optimal graph clustering by maximizing the ratio association measure, a criterion that formally defines the quality of clustering in this context:

$$KKM(C) = 2(n-k) - \sum_{i=1}^{k} \frac{\mathscr{E}_{c_k}^{in}}{\mid c_k \mid} \tag{3.25}$$

Community score: The Score Function, a community quality measure that maximizes intra-community edges in C, was proposed and is defined as[128]:

$$Score(C) = \frac{\sum_{i \in C} \left(\frac{1}{|C|} \sum_{i \in C} A_{ij} \right)^r}{\mid C \mid} \times \sum_{i,j \in C} A_{ij} \tag{3.26}$$

As a resolution parameter, r (a positive real number) regulates community size. The expression $\frac{1}{|C|} \sum_{i \in C} A_{ij}$quantifies the average fraction of connections from node i to other nodes in C, whereas $\sum i, j \in C A_{ij}$ corresponds to twice the intra-community edges.

The community score of divisions of the network can be define as follow :

$$CS(C) = \sum_{C=i}^{k} Score(C_i) \tag{3.27}$$

Intra-score: Measures how tightly connected nodes are within a community. Higher values indicate stronger internal cohesion [152].

$$IntraS(C) = \sum_{i=1}^{k} \frac{\mathscr{E}_{c_k}^{in}}{\mid c_k \mid} \tag{3.28}$$

3.3.2.3 Score function-based external connectivity

Ratio cut: Minimizing the ratio cut (the inter-community edge count) often partitions networks into large communities with sparse connections to the rest of the network.[171]

$$RC(C) = \sum_{i=1}^{k} \frac{\mathscr{E}_{c_k}^{out}}{\mid c_k \mid} \tag{3.29}$$

To prevent bias toward trivial communities (e.g., singletons), normalized ratio cut was proposed[172][153].

$$NRC(C) = \sum_{i=1}^{k} \frac{\mathscr{E}_{c_k}^{out}}{\mathscr{D} c_k} \tag{3.30}$$

External density: External density, represented as $EXD(C)$, measures the community C's external connection density and is defined by:

$$EXD(C) = \sum_{i=1}^{k} \frac{\mathscr{E}_{c_k}^{out}}{\mid c_k \mid (n - \mid c_k \mid)} \tag{3.31}$$

3.3.2.4 Combined score function that incorporates both internal and external connectivity

Community fitness: A measure to identify communities involves maximizing a node-based fitness function. This ratio metric divides intra-community edges by the total edges (internal + external), with community size tuned by positive real-valued parameter α.[89].

$$CF(C) = \sum_{i=1}^{k} \frac{\mathscr{E}_{c_k}^{in}}{(\mathscr{E}_{c_k}^{in} + \mathscr{E}_{c_k}^{out})^{\alpha}} \tag{3.32}$$

Conductance: This metric calculates the fraction of edges incident to the cluster that are inter-community connections [172][153].

$$Conductance(C) = \sum_{i=1}^{k} \frac{\mathscr{E}_{c_k}^{out}}{2\mathscr{D}c_k + \mathscr{E}_{c_k}^{out}} \tag{3.33}$$

Inter-score: Measures how separated different communities are from one another. Lower values indicate better separation[152].

$$InterS(C) = \sum_{i=1}^{k} \left[\frac{\mathscr{E}_{c_k}}{\mid c_k \mid} \right]^2 \tag{3.34}$$

The central goal of most clustering methods, whether stated directly or not, is to find clusters that are tightly knit internally (high IDC) but well-separated from each other (low $EXD(C)$). One straightforward approach to achieve this is to maximize the total sum of ($ID(C)$ - $EXD(C)$) for all clusters in the partitioning[54].

$$Sdf(C) = \sum_{i=1}^{k} ID(C_k) - EXD(C_k) \tag{3.35}$$

3.3.2.5 Score function based on similarity measure

This approach identifies communities with peak group similarity density in the network[13], formally expressed as:

$$\mathsf{SB}(C) = (1 + \frac{1}{1+\sqrt{k}}) \prod_{i=1}^{k} \mathsf{SD}_{Ci} \tag{3.36}$$

The adjustment factor $(1+\frac{1}{1+\sqrt{k}})$ is introduced to prevent nodes from diverging from a community that exceeds its expected size. Here, SD_{Ci} denotes the similarity density of the $i-th$ community, defined as: $\mathrm{SD}_{Ci} = \sum_{v,u\in C_i} S(v,u)$ where the sum represents the total similarity measure between all pairs of nodes within the community. The similarity measure $\mathrm{S}(v,u)$ of each pair(v,u) is computed using the following equations:

The Jaccard similarity coefficient: This metric quantifies vertex similarity by dividing shared neighbors by the combined unique neighbors of u and v[125]. Its value always lies in [0,1] and can be formalized as:

$$Jacard(v,u) = \frac{|N_u \cap N_v|}{|N_u \cup N_v|} \tag{3.37}$$

The Simpson similarity index: Unlike the Jaccard similarity index, which normalizes by the union of neighbors, the Simpson index normalizes by the smaller of the two node degrees. This measure is equivalent to the topological overlap coefficient, as noted in[72]. Formally, for two nodes v and u, it is expressed as:

$$Simpson(v,u) = \frac{|N_u \cap N_v|}{min\{d_u|,|d_v\}} \tag{3.38}$$

The Geometric similarity index: This index quantifies node similarity by squaring the count of shared neighbors and normalizing by the product of the nodes' degrees [125].

$$Geomiteric(v,u) = \frac{|N_u \cap N_v|^2}{d_u \times d_v} \tag{3.39}$$

The cosine similarity index: This metric is defined as the ratio of shared neighbors between two nodes to the geometric mean of their degrees.[67]. Formally, for nodes v and u, the cosine similarity is expressed as:

$$Cosine(v,u) = \frac{|N_u \cap N_v|}{\sqrt{|N_u| \times |N_v|}} \tag{3.40}$$

The Sørensen similarity index: This metric often used in ecological studies to quantify species similarity [125]. It is a measure analogous to the Jaccard index but adapted for social network analysis. The Sørensen index normalizes by the sum of the degrees of the two nodes. It is defined as:

$$Sorenson(v,u) = \frac{|N_u \cap N_v| \times 2}{|N_u| + |N_v|} \tag{3.41}$$

3.3.3 Evaluation of community score function

In [172], Yang and Leskovec proposed an axiomatic framework to evaluate the strength of community scoring functions. This framework leverages four goodness-of-community metrics: The ratio-cut $RC(C)$(see eq 3.29) and , the density $ID(C)$(see eq3.22), the $Conductance(C)$ (see eq3.33), and the clustering coefficient CC. A community with a high score in this framework is expected to exhibit strong performance in one or more of these metrics. These metrics collectively illuminate distinct aspects of network community structure, such as boundary clarity (separability), internal connectivity (density and cohesiveness), and local subgroup organization (clustering coefficient).

- Separability: The principle that effective communities exhibit minimal connections to external network nodes, quantified by the ratio cut metric $RC(C)$.
- Density: Effective communities exhibit high internal connectivity, quantified by metric $ID(C)$ as dense edge concentration among constituent nodes.
- Cohesiveness: This principle characterizes community structural integrity through uniform internal connectivity, where robust communities resist fragmentation. The property is quantified by the conductance metric $Conductance(C)$, which measures resistance to subdivision.
- Clustering coefficient CC : This principle assumes that network communities emerge from locally inhomogeneous edge distributions, as nodes sharing common neighbors exhibit a higher likelihood of mutual connectionsa phenomenon often linked to triadic closure in social networks using CC metric [170].

The clustering coefficient of node v is defined as follows:

$$CC_v = 2 \times \frac{|\, N_v \,|}{d_v(d_v - 1)} \tag{3.42}$$

3.3.4 Internal approaches based on quantitative evaluation:

Directly ranking different community detection approaches is challenging because they often rely on divergent definitions of what constitutes a community and employ distinct methodological frameworks. To overcome this limitation, many studies advocate for quantitative evaluation. After characterizing the communities identified by each method, these works prioritize metrics such as algorithmic complexity [140], execution time [139], scalability [53], and specific analytical capabilities (e.g., identifying life-cycle events [107]).

Furthermore, internal evaluation techniques are frequently employed to complement external validation, as demonstrated in [139, 160]. This reliance on multiple evaluation strategies arises primarily from the intrinsically ill-posed nature of the community detection problem itself.

3.4 Benchmarking and datasets:

A rigorous evaluation of algorithmic effectiveness necessitates comparison on both real-world datasets with established community structures and standardized synthetic benchmarks. To this end, two benchmark categories are employed: open and closed [79]. Open benchmarks rely on networks possessing verified ground-truth communities, while closed benchmarks utilize those without. This dual approach furnishes a consistent framework for evaluating the performance and accuracy of algorithms across varied network typologies, including static, dynamic, and multiplex.

3.4.1 Real-world dataset:

Table 3.1 summarizes key characteristics (model type, benchmark category, and node overlap) of the analyzed real networks and classifies them into three size-based classes:

- BSRP: Benchmark sets of small-scale real-world problems.
- BMRP: Benchmark sets of moderate-scale real-world problems .
- BLRP: Benchmark sets of large-scale real-world problems.

3.4.1.1 Statics networks:

Several notable social networks with established ground-truth partitions are frequently studied in the literature. These include Zachary's karate club network[178], the Bottlenose Dolphins network [103], Division I-A college football games network [59],Political Books Network[115], among others. A separate collection of real-world networks lacking publicly available ground-truth partitions also exists. Examples include the Train Bombing network, Slovene Parliamentary Party (SPP) network [52], the Windsurfers network, the Taro Exchange network [88], jazz musician collaborations [60], Netscience co-authorship [115], PGP network [19], DBLP co-authorship, and Amazon product networks [92].

3.4.1.2 Dynamics networks:

Below, we describe six authentic dynamic networks:

1. Mobile telephone communications (Cellphone);
2. Enron electronic correspondence;
3. DBLP collaboration network
4. Facebook network
5. Flickr network
6. Youtube network

Cellphone network: This dynamic network, sourced from VAST 2008 Mini Challenge 3, models call interactions among members of the fictional Paraiso movement (June 2006, 10-day period). Nodes represent participants; edges encode bidirectional calls [71].

Flickr network::This dataset (snapshot: January 9, 2007) models a social network with 1.8+ million nodes (users) and 22 million timestamped edges (links), where timestamps indicate exact connection creation times. [109].

Enron electronic correspondence network: This dataset comprises emails from the U.S. corporation Enron spanning 19992002 [81].

DBLP collaboration network: We construct a dynamic co-authorship network using DBLP data from 20122016, where: Each year represents a discrete time step; Nodes denote authors ; Edges signify collaborations in co-authored publications [8].

Facebook network: Models 52 % of New Orleans-region Facebook users (9/26/20061/22/2009). Nodes represent users; timestamped edges indicate friendship establishment events.[165]

YouTube network: Captured January 15, 2007, this temporal network contains 1.1M+ users and 4.9M timestamped edges recording subscription events [110].

3.4.1.3 Multiplex networks:

AUCS Network: This multiplex network captures five online/offline relationship types among Aarhus University employees: Facebook connections, leisure interactions, work collaborations, coauthorships, and lunch partnerships [141].

FF-TW-YT Network: This anonymized multiplex network integrates user relationships across three connected social platforms: FriendFeed (FF), Twitter (TW), and YouTube (YT). It captures cross-platform interactions among users who maintained linked accounts on these services[35].

FriendFeed-IT Network: A multiplex network capturing anonymized Italian user interactions on FriendFeed, including comments, likes and follows, with verified Italian language identification [35].

3.4.2 Synthetic Networks

The primary goal of Synthetic Network Generators (SNGs) is to model the fundamental characteristics underlying real-world phenomena and replicate them when generating synthetic networks, adhering to a limited set of straightforward constraints[138]. SNGs enable rigorous algorithm evaluation by generating benchmark datasets with controlled community structures.:

- Controlled Testing: SNGs allow precise adjustment of network features (e.g., size, density).
- Ground-Truth Validation: Certain generators inherently produce a ground-truth partition of the network. This enables direct comparison with the partition generated by the algorithm under test.

Several synthetic generators have been proposed; we mention the most widely used synthetic networks in the literature.

3.4.2.1 Synthetic Statics networks:

1. **Lancichinetti-Fortunato-Radicchi (LFR) Benchmarks:** Formulated by Lancichinetti-Fortunato, and Radicchi [90], LFR benchmarks provide a more accurate representation of real-world networks by incorporating heterogeneity in both node degree and community size. These networks also follow a power-law distribution, governed by exponents τ_1 and τ_2. A key parameter is the mixing parameter μ, which controls the balance between a node's internal and external connections. The specifications for these synthetic networks include the number of nodes (N), average degree (k_{avg}), maximum degree (k_{max}), and minimum and maximum community sizes ($MinC$ and $MaxC$).
2. **Girvan-Newman (GN) Benchmarks:** These synthetic networks contain 128 nodes partitioned uniformly into four communities, each comprising 32 nodes [116].

3.4.2.2 Synthetic Multiplex networks:

LFRm Benchmarks: Brodka and Grecki [28] proposed the mLFR benchmark a synthetic multiplex network generator extending the classical LFR model.

3.4.2.3 Synthetic Dynamics networks:

SYNFIX networks: Lin et al. [97] introduced a modified Girvan-Newman (GN) benchmark to evaluate FaceNet architecture. They incorporated network dynamics by generating sequential graph snapshots, with transitions between snapshots.

Dynamic LFR networks: Greene et al. [65] proposed a dynamic LFR benchmark simulating community evolution. The model initializes a static network with planted communities, then stochastically migrates 20 % of nodes between communities to emulate membership transitions.

RDyn networks: Rossetti [137] introduced RDyn, a novel benchmark specifically designed for Dynamic Community Detection (DCD). It generates evolving network topologies in both Snapshot Network (SN) and Temporal Network (TN) representations, along with time-evolving ground-truth communities.

Table 3.1: Comprehensive Benchmark Datasets for Community Detection Research

Dataset	Type	Nodes	Edges	Size Class	Key Characteristics	Reference
Static Networks with Ground Truth						
Zachary's Karate Club	Undirected	34	78	BSRP	Faction split in university club	[178]
Bottlenose Dolphins	Undirected	62	159	BSRP	Dolphin social interactions	[103]
College Football	Undirected	115	613	BSRP	Conference divisions	[59]
Political Books	Undirected	105	441	BSRP	Political alignment patterns	[115]
Static Networks without Ground Truth						
Train Bombing	Weighted	-	-	BSRP	Terrorist association network	[88]
Slovene Parliamentary Party	Signed	-	-	BSRP	Political alliance relationships	[52]
Jazz Musicians	Undirected	198	2,742	BSRP	Collaboration network	[60]
PGP Web of Trust	Undirected	39,796	-	BMRP	Cryptographic trust relationships	[19]
Amazon Products	Undirected	334,863	-	BLRP	Product co-purchasing network	[92]
Dynamic Networks						
Cellphone	Temporal	400	9,352	BMRP	10-day call interactions	[71]
Enron Emails	Temporal	184	150,000+	BMRP	Corporate email exchanges	[109]
Flickr	Temporal	1.8M	22M	BLRP	Social connections (Jan 2007)	[109]
DBLP Co-authorship	Snapshot	1.2M	5.4M	BLRP	Yearly collaborations (2012-2016)	[8]
Facebook NOLA	Temporal	63,731	1.5M	BLRP	Friendship evolution (2006-2009)	[165]
YouTube	Temporal	1.1M	4.9M	BLRP	Subscription network (Jan 2007)	[110]
Multiplex Networks						
AUCS	Multiplex	61	-	BSRP	5 relationship layers	[141]
FF-TW-YT	Multiplex	5,000+	-	BMRP	Cross-platform interactions	[35]
FriendFeed-IT	Multiplex	1,000+	-	BLRP	Italian user interactions	[35]
Synthetic Network Generators						
GN Benchmark	Static	128	-	-	4 equal communities	[116]
LFR Benchmark	Static	Configurable	-	-	Power-law degree distribution	[90]
mLFR	Multiplex	Configurable	-	-	Multilayer extension of LFR	[28]
SYNFIX	Dynamic	Configurable	-	-	Gradual community evolution	[97]

3.5 Taxonomy of Community Detection Methods

3.5.1 Foundational Classifications

Community detection methodologies can be categorized through multiple conceptual lenses. We analyze optimization-driven approaches across network types, static or dynamic, and present a unified taxonomy grounded in algorithmic strategies. This is followed by critical analysis of strengths/limitations.

3.5.2 Existing Taxonomies

3.5.2.1 Algorithmic Paradigm Classification (Attea et al.)

Attea et al. [11] classify community detection into four primary paradigms. Heuristic-based methods (HCD) employ rule-driven approximations for efficient solutions. Metaheuristic approaches (MCD) leverage stochastic optimization frameworks like evolutionary algorithms. Hybrid metaheuristic methods (HMCD) synergistically combine HCD and MCD through three cooperation strategies: low-level teamwork (LT) embeds heuristic components within metaheuristic frameworks; high-level relay (HR) executes sequential HCD $\rightarrow$ MCD $\rightarrow$ HCD pipelines; and high-level teamwork (HT) coordinates parallel metaheuristic executions. The hyper-heuristic paradigm (HH) operates at a higher abstraction level, dynamically selecting or generating suitable heuristics/metaheuristics based on problem characteristics. This taxonomy systematically categorizes solution strategies by their fundamental operational mechanisms and integration methodologies.

3.5.2.2 Dynamic community detection taxonomy (Rossetti et al.)

Rossetti [138] categorizes dynamic community detection methods by their temporal dependency approach. Instant-Optimal methods optimize communities per snapshot through iterative similarity matching, core-node continuity tracking, or multi-step temporal alignment. Temporal Trade-off approaches balance current structural optimality with historical consistency using techniques like global optimization seeding, rule-based incremental updates, multi-objective optimization (jointly maximizing structural quality and temporal coherence), or historical network smoothing. Cross-Time methods identify unified communities across the entire timeline through four evolutionary models: (1) fixed memberships and properties, (2) fixed memberships with evolving properties, (3) evolving memberships with fixed properties, or (4) fully evolving memberships and properties. This taxonomy systematically addresses the fundamental trade-off between snapshot-level optimization and temporal continuity preservation in dynamic network analysis.

3.5.2.3 Dynamic community detection evolution tracking (Dakiche et al.)

Dakiche [41] systematizes evolutionary community tracking into four methodological approaches. The independent detection + matching paradigm first identifies communities per snapshot before establishing cross-temporal correspondences through similarity metrics. Dependent detection incorporates temporal constraints directly into the discovery process, using historical state information as initialization or regularization. Simultaneous multi-snapshot optimization jointly analyzes multiple timesteps through global objective functions that maximize both structural quality and inter-snapshot consistency. Finally, online temporal network processing handles streaming data via incremental algorithms that update communities in real-time as new interactions arrive, typically employing window-based models or decay mechanisms. This taxonomy highlights fundamental methodological divisions in how algorithms incorporate temporal dimensionranging from post-hoc matching to fully integrated processingproviding a framework for evaluating trade-offs between computational efficiency and evolutionary coherence preservation in dynamic networks.

3.5.3 The novel taxonomy for community detection based optimization:

In this subsection, we present our taxonomy for community detection optimization approaches. We first outline the rationale behind the classification and then discuss the advantages and limitations of each class.

Our taxonomy spans the network models mentioned earlier, including static networks (directed, weighted, signed, multiplex) and dynamic networks(snapshots and dynamic) as shown in Fig 3.2.

3.5.3.1 Heuristic based community discovery:

A heuristic algorithm is characterized as a collection of deterministic, rule-based algorithms aimed at iteratively discovering locally optimal candidate solutions through the integration of domain-specific knowledge. It operates by producing local variations of a specified solution and subsequently employing a particular criterion to assess the potential of neighboring solutions. A heuristic community detection algorithm HCD is an iterative transformation function represented by the tuple $\mathsf{HCD} = (\Omega, C, \mathrm{h}, \mathscr{F}, \mathsf{T})$. The algorithm functions via an iterative learning process, commencing with an initial dividing solution C within the search space Ω, which includes all potential solutions. It subsequently enhances the solution through the iterative application of three fundamental components:

- A local variation operator h to generate modified solutions

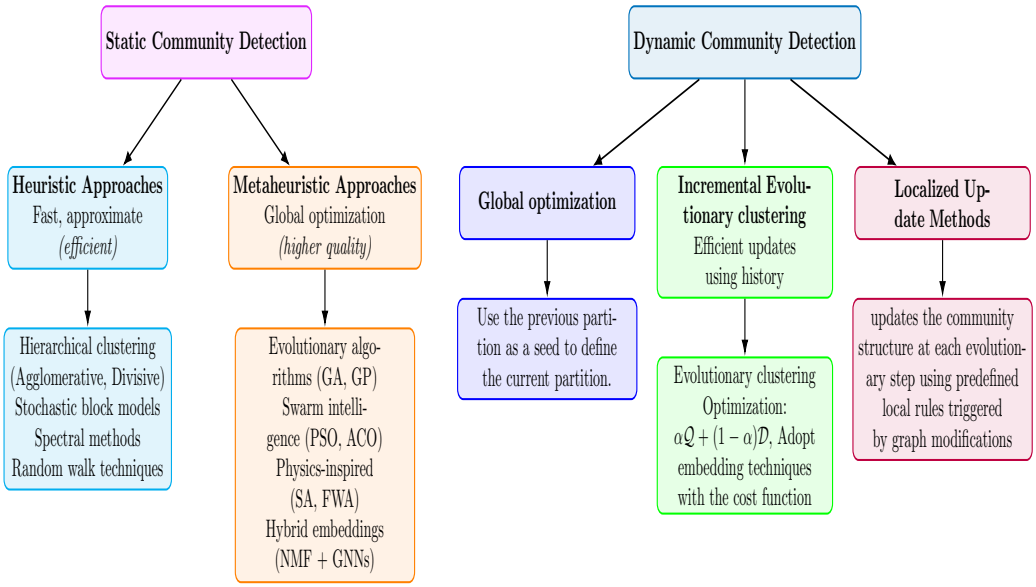


Figure 3.2: **A Taxonomy of Optimization Approaches for Community Detection.**

- An evaluation function $\mathscr{F}$ to assess solution quality
- A termination criterion T to halt the process once predefined conditions are met.

Generally, existing HCD approaches are classified into the following categories [54]:

- Hierarchical methods (including agglomerative and divisive algorithms)
- Stochastic block models
- Spectral clustering
- Random walk-based techniques

Advantages and Drawbacks of heuristics for Community Detection:

Advantages:

1. Speed and Efficiency: Heuristics have the potential to substantially decrease the time necessary to identify a solution. The problem-solving process is simplified,

which is particularly beneficial in dynamic environments where time is a critical factor, enabling speedier decision-making.

2. Simplicity: In comparison to more intricate algorithms, heuristic methods are frequently simpler to implement and comprehend.

3. Flexibility: Heuristics can be customised to address a wide range of issues and can be adjusted to meet the unique needs or context of the situation. This adaptability enables a more extensive application in a variety of domains.

Drawbacks:

1. Optimality and validity are among the primary disadvantages of heuristics, as they do not ensure an optimal solution. The solutions they offer may be suboptimal, which can be problematic in situations where the best outcome is essential.

3.5.3.2 Metaheuristics based community discovery:

Metaheuristics are high-level optimization strategies inspired by natural phenomena or abstract conceptual processes [17], widely employed for community detection in complex networks. These algorithms are particularly valuable for addressing large-scale or NP-hard optimization problems where exact methods become computationally intractable.

In the context of community detection (CD), numerous metaheuristic algorithmsoften termed Metaheuristic Community Detection (MCD) algorithmshave been proposed. Generally, an MCD algorithm employs strategies inspired by natural or conceptual processes (e.g., evolutionary, swarm-based, or physical systems) to iteratively explore and refine community structures.

Formally, an MCD can be defined as an iterated transformation function: MCD = $(\Omega, \mathscr{F})$ where:

- A set of variables: IP = $\{p_1, \ldots, p_n\}$
- A variable domain: $D_1, \ldots, D_n$
- $\Omega = \{\text{IP} = \{p_1, \ldots, p_n\} \mid p_i \in D_i\}$, such that satisfies all the constraints
- $\mathscr{F}$ is the community structure evaluation function. $\mathscr{F} : D_1 \times \cdots \times D_n \to \mathbb{R}^+$

The algorithm operates on an initial population IP = $p_1, \ldots, p_n$ of n encoded solutions within the search space Ω. Despite belonging to different families, most MCD algorithms follow a common strategy: formulating CD as either a single-objective optimization (guided by $\mathscr{F}$) or a multi-objective optimization, where $\mathscr{F}$ may represent one or more evaluation metrics (e.g., modularity, conductance, or NMI) 3.3.

Recent advancements have focused on hybridizationintegrating complementary components like machine learning, quantum-inspired operators, or problem-specific heuristics into metaheuristic frameworks to enhance performance.

The literature emphasizes that the effectiveness of metaheuristic community detection approaches [121, 129] hinges on how well the objective function $\mathscr{F}$ captures the notion of a "good community structure." When aligned with this structural definition, the algorithms search process becomes purposeful, effectively leveraging core strategies like intensification, diversification, and learning (IDL) to guide the optimization. Research highlights that successful MCD algorithms depend on these three core strategies [17, 62]:

- **Intensification**: Refers to focusing the search by exploiting the area nearby current good solutions.
- **Diversification**: Exploring new regions of the search space.
- **Learning**: Adapting based on prior results.

These IDL components ensure the algorithm balances precision with adaptability, enabling robust community detection across diverse network configurations. Conversely, if the objective function is poorly defined, even sophisticated IDL strategies may fail to deliver meaningful results, rendering the algorithm inefficient despite its computational effort.

3.5.3.3 Hybrid metaheuristics HMCD:

HMCD are computational frameworks designed for the implicit or explicit integration of metaheuristics with other heuristic or metaheuristic techniques[161]. The primary motivation behind their development stems from the recognition that hybridization is crucial not only for leveraging the strengths of individual algorithms but also for overcoming the limitations,such as premature convergence or poor scalability in standalone methods, inherent in their pure counterparts [11].

Advantages and Drawbacks of Metaheuristics for Community Detection:

Advantages:

1. Efficiency in Complex Problems: Metaheuristics, including Genetic Algorithms (GAs) and Particle Swarm Optimization (PSO), are particularly effective for addressing complex optimization challenges, such as community detection. They possess the ability to efficiently investigate vast solution spaces, which is essential due to the computational requirements associated with identifying optimal network partitions.

2. Adaptability: A multitude of metaheuristic approaches can be modified to accommodate various network and community structures. For example, the Fireworks Algorithm and Sine-Cosine meta-heuristic have been specifically designed to facilitate community detection, thereby demonstrating their adaptability.

3. Hybridization: Metaheuristics can improve their efficacy by incorporating a variety of search strategies and operators. The integration of various search operators and similarity metrics has been demonstrated to produce competitive outcomes in community detection tasks, for instance.

4. Robustness: These methods frequently offer resilient solutions to a variety of community detection issues. Although no single algorithm is superior, the high variability in algorithm performance suggests that numerous methods can perform well under varying conditions.

Drawbacks:

1. Despite the potential for metaheuristics to be efficient, they may still necessitate substantial computational resources, particularly when applied to large networks. The requirement for extensive parameter tuning and multiple runs to attain optimal results can be time-consuming.

2. The design and implementation of metaheuristic algorithms can be complex, necessitating a profound comprehension of both the algorithm and the specific problem domain. Practitioners who lack the requisite expertise may encounter this complexity as an obstacle.

3.5.3.4 Incremental approaches:

Incremental algorithms process dynamic network changes step-by-step, leveraging both the current network state and previously identified community structures from the prior iteration to update community assignments [139][179]. This iterative, state-aware framework is characteristic of Dynamic Community Detection (DCD) algorithms, which prioritize efficiency by minimizing redundant recomputation while preserving temporal dependencies in evolving networks(see Fig 3.3):

- Initialization: Detect communities in the networks initial state
- Update: At each timestep t, detect communities in the evolving network by analyzing the current network state (G_t) alongside historical data (e.g., prior community assignments C_{t-1}) to ensure temporal continuity.

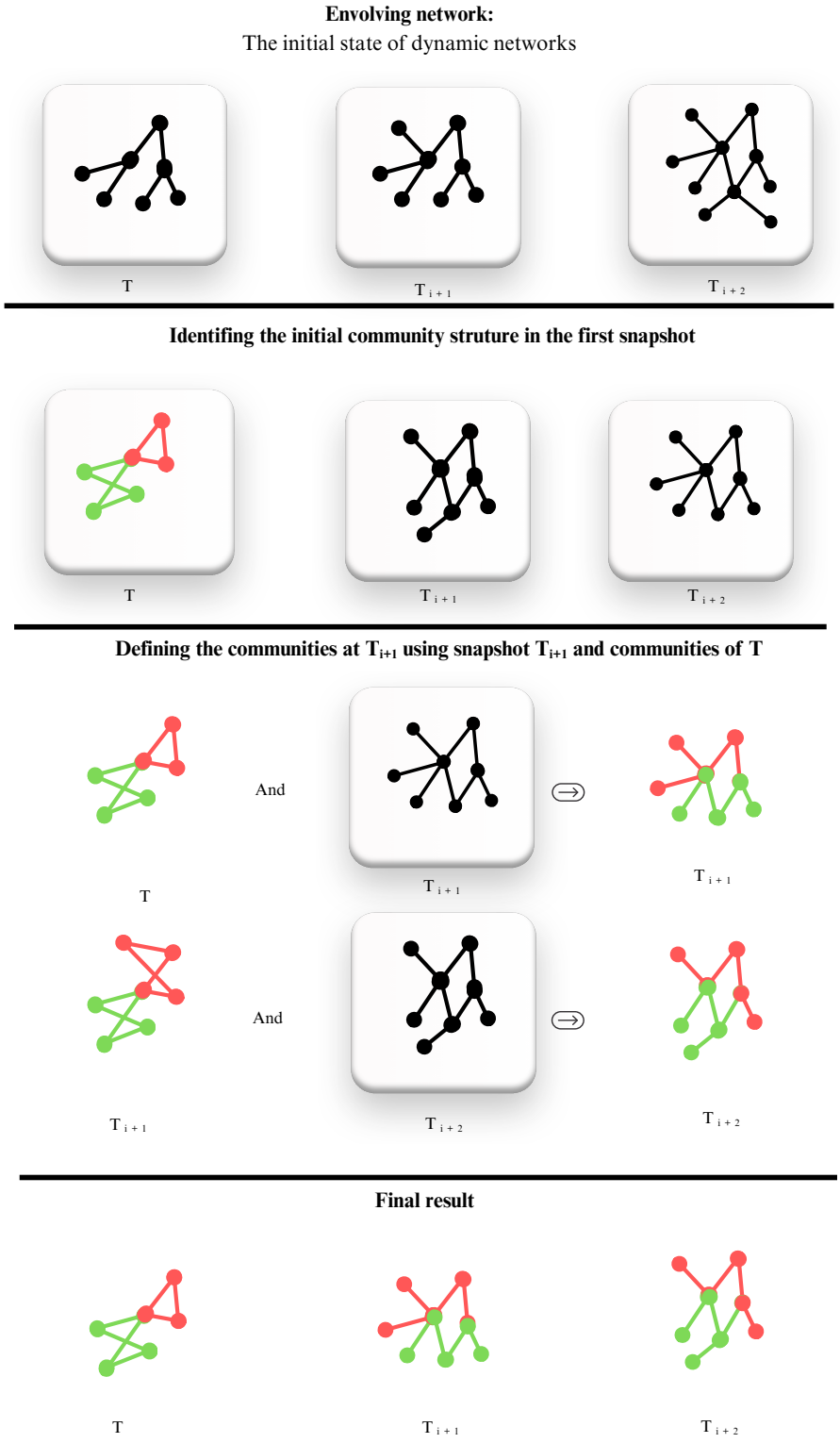


Figure 3.3: **Incremental approach process.**

Incremental approaches leveraging optimization algorithms can be categorized into three main subclasses:

1. **Global Optimization**: Methods in this subclass initialize the global optimization process at timestep $t+1$ by using the partition from t as a seed. These approaches optimize a global quality function (e.g., modularity or conductance) akin to static community detection (CD). The optimization heuristics employed may either mirror those used in static CD or incorporate dynamic adaptations tailored to evolving networks.

2. **Incremental evolutionary clustering**:(Chakrabarti et al. [86]) This approach generates temporal cluster sequences through a smoothing framework that balances current-period accuracy with historical consistency. The method formalizes the temporal smoothness hypothesis: community structures exhibit inertia and resist abrupt changes between consecutive time intervals. This principle enables gradual structural evolution by minimizing disruptive partition shifts during sequential clustering. Smoothing is accomplished by balancing two distinct objectives; the first objective, termed snapshot quality, indicates that the clustering results should accurately represent the network's community structure at the present time step. The second objective relates to time cost, indicating that each clustering outcome should avoid significant variations between consecutive time steps. According to Paul et al.'s comprehensive analysis [126], most methodological frameworks for dynamic community detection employ multi-objective optimization and evolutionary algorithms. These approaches fundamentally balance two competing objectives:

   - Temporal smoothness: Ensure that the structure of the community at time t reflects a natural evolution from the partition identified at $t-1$.
   - Structural Quality: Maximizing partition quality (e.g., modularity) at each snapshot To achieve this balance, the method optimizes a composite objective function $cost$ that integrates both temporal smoothness CT and structural metrics CS.

$$cost = c = \alpha CS + (1-\alpha)CT \tag{3.43}$$

3. **Localized update methods** : This approach incrementally updates the community structure at each evolutionary step using predefined local rules triggered by graph modifications (e.g., edge additions/removals). The method prioritizes computational efficiency, aiming to achieve modularity scores comparable to no-smoothing baselines while avoiding costly full recomputation of communities at every timestep.

Advantages and Drawbacks of incremental approaches:

Advantages:

1. The majority of methods in this class utilize the current community structure at step $t-1$ to expedite the community detection at t.

2. The instability problem can be addressed through incremental approaches, which employ the smoothness technique described in the previous subsection (see Section 6.4).

3. Universal Applicability: The method is applicable to all dynamic network models, including snapshot-based representations and temporal networks with continuous-time interactions.

Drawbacks:

1. Parallelization Limitation: Parallelizing the community detection process is often impractical due to its inherently sequential dependency, each subsequent phase relies on the community assignments generated in the preceding iteration. This interdependency creates a critical path that hinders concurrent computation, limiting scalability for large-scale dynamic networks.

2. Temporal Coherence Challenge in Dynamic Communities: A critical limitation in dynamic community detection lies in maintaining temporal consistencythe ability to preserve meaningful community structures across time. Methods employing iterative incremental approaches risk cumulative error propagation due to their reliance on prior state dependencies. Over successive iterations, this can lead to significant divergence between dynamically detected communities and the system's current state, as benchmarked against static network snapshots.

3.5.3.5 Embedding-Metaheuristic Integration

Node embeddings $f : \mathcal{V} \rightarrow \mathbb{R}^d$ fundamentally transform metaheuristic community detection (MCD) through three synergistic mechanisms. By preserving topological properties in continuous latent spaces, embeddings convert discrete graph optimization into geometric manifold learning. This enables efficient geometric operations (vector clustering, distance-based crossover) that replace costly graph traversals. Furthermore, embeddings facilitate cross-network generalization through alignment techniques that transfer structural patterns between domains.

This integration directly addresses core MCD limitations: dimensionality reduction accelerates convergence, continuous representations enable gradient-based operators, and transfer learning mitigates cold-start problems. The resulting framework maintains

detection quality while scaling to dynamic topologies through structural regularization in the embedding space.

Integration Strategies Embedding techniques transform discrete graphs into continuous spaces through distinct approaches. Matrix factorization ($\min_{\mathbf{U},\mathbf{V}} \|\mathbf{A} - \mathbf{U}\mathbf{V}^\top\|_F + \mathscr{R}$) preserves global structure through low-rank approximations. Random walk methods (DeepWalk, node2vec) capture higher-order neighborhood proximities through simulated traversals. Graph neural networks leverage feature propagation ($\mathbf{Z}^{(l+1)} = \sigma(\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{Z}^{(l)}\mathbf{W}^{(l)})$ to handle attributed and dynamic graphs.

Collectively, these methods convert sparse graph data into dense geometric manifolds suitable for efficient MCD implementation. Matrix methods excel at global structure preservation, random walks capture local proximities, and GNNs adapt to complex graph types, enabling community detection through manifold learning in $\mathbb{R}^d$ with polynomial rather than combinatorial complexity.

3.6 Community detection based optimization methods

In this section, we provide a comprehensive analysis of recent community detection methods. Table 3.2 summarizes the most important methods, presenting their complexity, key features, and type of hybridization.

3.6.1 Static community detection algorithms

3.6.1.1 Heuristic methods

Greedy heuristics represent a direct and efficient method for addressing combinatorial optimization issues, as they rapidly generate workable solutions gradually by selecting the most advantageous irreversible local option at each stage.

Newman (2004) introduced the first modularity-based greedy algorithm for community detection [114]. This agglomerative hierarchical method incrementally merges nodes to optimize modularity, with worst-case time complexity $O((m+n)n)$ for graphs containing n nodes and m edges.

Concurrently, Clauset et al. [38] reduced the time complexity to $O(md\log n)$ where d is the dendrogram depth using advanced data structures like max-heaps for efficient storage and updating of modularity gains. This optimization significantly improved algorithm speed and efficacy. Key advantages include automatic community count detection, eliminating prior specification requirements. However, the greedy approach risks convergence to local optima, potentially yielding suboptimal partitions.

The Louvain method [16] enhances modularity optimization through an iterative two-phase process with approximate $O(m)$ complexity, though efficacy depends on node

visitation order. Initially, each node is assigned to its own community; nodes are then sequentially moved to adjacent communities only if a positive modularity gain ($\Delta Q > 0$) is achieved. This local optimization repeats until modularity converges. The second phase aggregates communities into supernodes, rebuilding the network with these as vertices. Both phases iterate until global modularity maximization is achieved, enabling near-linear scalability in edges m that makes the method suitable for large-scale networks.

Liu and Murata [100] developed LPAm, a label propagation algorithm optimizing modularity, and enhanced it through integration with a multistep greedy agglomerative algorithm (MSG) that simultaneously merges multiple community pairs. This advanced version, LPAm+, demonstrates superior modularity optimization performance compared to the original LPAm.

Waltman et al. [166] introduced the Smart Local Moving (SLM) algorithm, a modularity-optimization method for community detection in large networks. While structurally similar to the Louvain method, SLM's key innovation lies in its recursive sub-network refinement: before graph aggregation, each existing community is treated as an independent sub-network where a local moving heuristic is applied. This heuristic 1) initializes all nodes within the sub-network as singleton communities, then 2) iteratively reassigns nodes to maximize modularity. The refined communities are then aggregated, with this process repeating until convergence, yielding higher-quality partitions than single-pass approaches through multi-resolution optimization.

Traag [162] proposed the Random Neighbor Move (RNM) modification to reduce the Louvain method's theoretical runtime from linear in edges $O(m)$ to $O(n \log k)$ (where k is the average degree). Instead of relocating nodes to the optimal adjacent community in the first phase, RNM moves nodes to a random adjacent community. This stochastic approach enhances solution space exploration but may marginally reduce partition quality compared to the deterministic Louvain algorithm.

Ozaki et al.[122] introduced the Louvain Prune algorithm to accelerate the Louvain method. By selectively evaluating neighboring communities during modularity optimization (via the Fast Local Move (FLM) operation) rather than exhaustively checking all neighbors, it reduces runtime complexity. Experiments demonstrate up to 10Œ speed improvement over standard Louvain.

The Leiden algorithm [163] substantially improves upon the Louvain method by addressing core limitations and integrating enhancements from SLM [166], FLM [122], and RNM [162]. Unlike Louvain, which can yield disconnected communities (multiple isolated subgraphs grouped as one), Leiden guarantees well-connected communities through a refinement phase. While maintaining comparable $O(m)$ complexity, Leiden's refinements accelerate convergence and produce superior partitions.

Multiplex networks heuristics Community detection heuristics for multiplex networks are categorized as global or local [106]. Global approaches include flattening, layer-by-layer, and multilayer methods. Flatteningthe simplest techniquemerges network layers into a single graph for standard community detection. Weighted variants assign edge weights reflecting original structural characteristics to improve fidelity. Despite enhanced noise robustness, flattening may: (i) only identify pillar communities (edges spanning multiple layers), and (ii) generate artificial communities from distributed cross-layer connections.

In contrast to flattening methods, layer-by-layer approaches apply conventional community detection independently to each network layer before integrating results. These include: association rule mining; consensus/ensemble-based detection (EMCD); and spectral methods like Principal Modularity Maximization (PMM) [106].

Multilayer approaches instead operate directly on the multiplex structure, categorized into four types: i) Density-based methods (ii) Random walks (e.g., LART, Infomap) (iii) Optimization techniques (e.g., Generalized Louvain) (iv) Label propagation.

The extension of single-layer community detection techniques to multilayer networks began with Mucha et al.'s pioneering study [112]. This work introduced a modularity definition for multilayer networks and yielded a generalized version of the single-layer Louvain method [16], implemented in the GenLouvain software package.

In this paper, Interdonato et la [74] introduces a novel framework for local community detection in multilayer networks. Our method maximizes the internal-to-external connection density ratio within communities, using similarity-driven relationships across layers, and incorporates a biasing scheme to identify communities with diverse layer coverag.

Bergermann et al.[14] propose two unsupervised methods for undirected multiplex networks, both modeled as energy functional gradient flows:

- MPBTV minimizes a balanced total variation functional (proven equivalent to multiplex modularity maximization)
- DGFM3 directly maximizes multiplex modularity. Their nonlinear matrix ODE formulations are efficiently solved using a graph-adapted Merriman-Bence-Osher (MBO) scheme, bridging theoretical foundations with practical computational efficiency.

3.6.1.2 Metaheuristic methods

Metaheuristic methods have substantially advanced solutions to complex graph-based combinatorial optimization problems through their global search capabilities. As pivotal tools in computational intelligence, they provide robust solutions for intricate network

analysis. Consequently, numerous metaheuristic strategies have emerged for community detection. This section critically examines prominent contemporary approaches in the field.

In 2008, Pizzuti[128] proposed a single-objective evolutionary algorithm (GA-Net) to address the community discovery problem. This approach aimed to overcome limitations of greedy search methods, which are prone to becoming trapped in local optima. Pizzuti employed a locus-based adjacency encoding scheme to represent the initial population and introduced the community score function (denoted as $CS(C)$ in Eq3.27, a quality metric designed to comprehensively evaluate community structures. The same single-objective evolutionary algorithm was then used to optimize $CS(C)$. This function incorporates a parameter r to regulate community size, emphasizing the importance of internal node degrees within a community to reinforce its structural significance.

Shi et al. [151] introduced the seminal multi-objective community detection (MOCD) method, decomposing modularity Q into two optimization components: intra-community strength (eq3.28) and inter-community strength (eq3.34). By simultaneously maximizing these objectives, MOCD generates diverse community configurations that explore trade-offs between internal cohesion and external separation. Crucially, since Q represents their sum, the partition maximizing Q necessarily resides in the Pareto-optimal set. When evaluated against three modularity-based benchmarks (two heuristics and one single-objective genetic algorithm), MOCD achieved superior accuracy - though further improvements in quality metrics and algorithmic efficiency remain warranted.

In [37], Cheng et al. developed a multi-objective evolutionary algorithm, termed the Local Information-based Multi-Objective Evolutionary Algorithm (LMOEA), to address the community detection problem. The method simultaneously optimizes two conflicting objectives: the ratio cut ($RC(C)$)3.29 and the neighborhood ratio association ($NRC(C)$), defined in Eq3.30. Minimizing $NRC(C)$ promotes partitions with densely connected internal communities, whereas $RC(C)$ aims to reduce inter-community connections. These objectives operate in tension, as optimizing one inherently compromises the other.

Cai et al.[30] presented a discrete particle swarm optimisation (DPSO) method for identifying communities in signed networks, in which the particle update mechanisms were reformulated to incorporate the structure of the network.

Naeni et al. [95] proposed MAGA-Net, a memetic algorithm combining adaptive mutation operators and competitive neighborhood dynamics within a multi-agent genetic framework to optimize modularity. Comparative evaluation shows significantly improved partition accuracy and faster convergence versus MA-Net, demonstrating enhanced performance in modularity-driven community detection.

In [144], the authors proposed a Clustering Coefficient-based Genetic Algorithm (CC-GA) for community detection. The algorithm employs the Clustering Coefficient

(CC)3.42 to initialize the population and uses modularity as its fitness function. CC-GA incorporates genetic operators such as uniform crossover and a newly proposed mutation mechanism.

Sanchez et al. [147] proposed a multi-start iterated greedy algorithm (MSIG) for community detection in large social networks. The method employs a Greedy Constructive Procedure (GCP) to generate an initial solution and reconstruct partial solutions during the reconstruction phase. GCP begins by selecting a seed node to form an initial community, then iteratively incorporates remaining nodeseither merging them into existing communities if modularity improves or creating new communities otherwise. This bottom-up aggregation continues until all nodes are assigned, progressively forming larger communities.

Hanzhang et al. [84] enhanced the iterated greedy metaheuristic framework through the Iterated Carousel Greedy (ICG) algorithm. ICG generates multiple solutions via iterative cycles of: (1) destruction (partial deconstruction), (2) carousel (problem-specific greedy heuristics), and (3) reconstruction (greedy solution rebuilding). Each iteration incorporates a local search refinement. Experiments confirm ICG's superiority over MSIG [147], DBA [156], and Louvain [16]

Furthermore, to diminish computational expenses and enhance the accuracy of MSIG [147], Li et al. [94] included the Louvain algorithm [16] into the IG framework. This integration enabled the Louvain approach to recover the partially affected solution throughout the reconstruction process. A local search approach was employed after reconstructing to improve the algorithm's efficacy.

In signed networks, Girdhar and Bharadwaj [58] extended the multi-objective framework by integrating a social-balancing component alongside the modularity and frustration objectives. These objectives were optimized using the NSGA-II algorithm, enhancing the models ability to address conflicting goals in community detection.

Liu et al. [99] proposed an Iterated Local Search (ILS) algorithm combining GCP initialization [147] and modularity-optimizing label propagation [100] for local search. This approach outperforms MAGA-Net [95], Louvain [16], and DBA [156] in community detection benchmarks.

Zhang et al. [184] developed WOCDA, a modularity-optimizing community detection algorithm using whale-inspired techniques (shrinking encircling, spiral updating, random search). While outperforming DPSO [30] and BA [68] in partition accuracy, its computational cost increases significantly with network size due to random search limitations.

Imtiaz et al. [73] introduced MLACO a multi-layer Ant Colony Optimization method for community detection in complex networks. Using ratio cut $RC(C)$ (Eq. 3.29) and kernel k-means $KKM(C)$ (Eq. 3.25) as objective functions, MLACO demonstrates effective

community identification across synthetic and real-world networks.

Saeid et al. [154] improved the Cuckoo Search Optimization (CSO) algorithm for community detection by integrating Genetic Algorithm (GA) operators to mitigate issues such as premature convergence and local optima. The hybrid approach leverages crossover and mutation operations to balance exploration and exploitation, coupled with dynamic population sizing to adaptively refine search efficiency. Modularity 3.15 and Normalized Mutual Information 3.8 were adopted as conflicting multi-objective functions to guide the optimization process.

Spampinato et al. [157] introduced opt-IA, an immune optimisation algorithm for community detection that maximizes modularity. Evaluated against 20 heuristics and metaheuristics (including one Hyper-Heuristic) on social and biological networks, opt-IA uniquely employs a purely stochastic search methodology with three operators: Static cloning: Generates new populations from highest-fitness solutions. Hypermutation: explores local search space neighborhoods. Stochastic ageing: eliminates solutions probabilistically to escape local optima. This random-search framework demonstrates competitive performance while avoiding deterministic components.

Kumar et al. [87] proposed a deep learning method for community detection combining stacked autoencoders and CSA-enhanced k-means. After constructing the graph's modularity matrix, stacked autoencoders compress it while preserving topological features. The reduced representation is clustered via k-means optimized with Crow Search Algorithm (CSA), which determines optimal initial centroids replacing random initialization to enhance accuracy and stability in community identification.

Reihanain et al [135]proposed an evolutionary algorithm called MOBBO-OCD, based on multi-objective biogeography-based optimization (BBO), identifies overlapping communities in social networks with node attributes by simultaneously optimizing two objectives: connection density and node-attribute similarity. The algorithm employs modularity and the SimAtt measure (a metric that quantifies attribute similarity within communities) as its dual objective functions. To encode and decode overlapping communities, the authors introduce OLAR (Overlapping Locus-based Adjacency Representation), an extended adjacency representation scheme. Within the evolutionary process of MOBBO-OCD, a rank-based migration operator, a novel two-phase mutation strategy, and a double-point crossover mechanism are applied to guide the population toward optimal solutions. These components leverage OLAR to efficiently explore the search space. For performance evaluation, the paper proposes α-SAEM, a new metric that assesses the quality of both overlapping and non-overlapping community partitions by integrating structural connectivity (linkage density) and node-attribute homogeneity. This metric addresses the limitations of existing measures by holistically evaluating communities through structural and attribute-based criteria.

The position update rule of the basic Coot bird optimization algorithm (COOT)lacks a balance between exploitation and exploration abilities. To address this problem, a new update mechanism is integrated into the basic COOT method to enhance its local and global search capabilities suggested by Aslan et al[6]. This improved version, referred to as the Modified COOT (MCOOT) method, introduces three key enhancements in its update mechanism: A subset of dimensions (ranging from 1 to the problems dimension size) is randomly selected for each coot individual. The selected dimensions are modified using a novel update rule to refine the search process. A genetic mutation operator is applied to the current coot position based on a predefined mutation probability, further boosting exploration capabilities.

Recently, This paper [105] introduces a hybrid meta-heuristic framework combining the Whale Optimization Algorithm (WOA), Salp Swarm Algorithm (SSA), and a novel Cloud Optimization (CO) algorithm. An adaptive hyperheuristic, based on a modified choice function, manages the optimization process to balance exploration-exploitation efficiently. Inspired by atmospheric particle dynamics, the CO component boosts population diversity, prevents premature convergence, and improves the framework's ability to model complex network topologies

Multiplex networks: In their work, Karimi et al. [77] developed a hybrid method integrating the Multi-Objective Evolutionary Algorithm with Decomposition (MOEA/D) with Tabu Search (TS) and the Clustering Coefficient (CC)3.42 metric to detect structurally similar communities in multiplex networks. Their approach operates in two key stages: The clustering coefficienta social network analysis metric measuring local connectivityis used to generate an initial population of high-quality candidate solutions. Tabu Searchs neighborhood exploration capabilities are embedded within the MOEA/D framework to efficiently navigate the solution space and identify Pareto-optimal solutions, balancing competing objectives.

Boukaben et al.[25] proposed the Smart Flattening-based Cuckoo Search Algorithm (SFCSA) for multiplex network community detection. This three-phase approach:(1) Generates fused embeddings using DeepWalk and node2vec to minimize information loss.(2) Constructs a flattened network via layer integration, with edge weights calculated from Pearson correlation coefficients. (3) Identifies optimal clusters through modularity-maximizing3.15 cuckoo search on embeddings by unifying topological representation (phases 1-2) and optimization (phase 3), SFCSA effectively reveals community structures while preserving multiplex relationships.

3.6.2 Dynamic community detection algorithms:

3.6.2.1 Incremental approaches:

Localized update methods: Di Zhuang et al.[185] introduced DynaMo, a dynamic community detection method designed to optimize modularity gains during community structure evolution. DynaMo represents network dynamics as a sequence of incremental updates, categorized into six types: Intra-community edge additions/weight increases, Inter-community edge additions/weight increases, Intra-community edge removals/weight decreases, Inter-community edge removals/weight decreases, Node additions, Node removals. The method operates in two phase: Construct a provisional community structure based on the incremental update and the previous community configuration; apply the local moving and aggregation phases of the Louvain algorithm [16] to the intermediate structure until modularity cannot be further improved. By focusing on localized updates, DynaMo achieves efficient modularity optimization without reprocessing the entire network.

In this paper[24], Fariza Bouhatem et al presented an innovative density-based method using dual optimisation to monitor community structures within expanding social networks. The proposed incremental algorithm exhibits relatively low complexity, making it efficient and fast. Designed for dynamic networks, which evolve through the addition of nodes and links, the method employs localised density optimisation to mitigate the resolution limit inherent in modularity optimisation approaches.

In [179], Neda Zarayeneh et al. introduced a fast, incremental method for dynamic community detection. Their key contribution is a general-purpose framework called $\Delta-$screening, which processes batches of updates to a graph (e.g., edge/node additions/removals) and identifies a subset of nodes requiring reevaluation for potential community reassignment. This framework is compatible with any modularity-driven community detection algorithm. To demonstrate its versatility, the authors integrated $\Delta-$screening into two widely-used community detection approachs(louvain and SLV), enhancing their adaptability to evolving networks.

Global optimization methods Bansla et al.[9] introduced fast community detection for dynamic complex networks FCDDCN for real-time community detection in dynamic networks. Leveraging CNMs initial community structure [38], it dynamically updates adjacency lists for edge modifications and optimizes modularity through heuristic search. This approach enables efficient iterative refinement of communities while adapting to network changes.

The method proposed by Shang et al.[148] identifies dynamic communities by maximizing modularity over time. It initializes communities in the first snapshot using the Louvain algorithm[16]. For subsequent network changes, edge-specific update rules

are dynamically applied to refine community assignments while preserving modularity. These rules adapt to the type of edge modification, categorized as: Intra-community edges, Inter-community edges, Semi-new edges, New edges. By tailoring updates to the nature of network adjustments, the method ensures efficient adaptation to evolving structures.

In this paper, Ranjkesh et al[132] proposed a memetic algorithm named RDMA-NET to enhance the resilience of communities in complex networks against attacks. The algorithm maximizes appropriate robustness criteria to improve the resilience of dynamically detected communities. It integrates evolutionary algorithms with community structure robustness metrics.

Incremental evolutionary clustering: Folino et Pizzuti [53]proposed a dynamic multi-objective genetic algorithms (DYNMOGA) use genetic algorithms to detect communities in dynamic networks by optimizing two objectives: NMI(eq 3.8) and modularity (eq3.15). NMI preserves temporal consistency with prior community structures, while modularity maximizes snapshot quality by assessing how well the current clustering represents the network. However, DYNMOGA suffers from a critical limitation: despite optimizing both objectives simultaneously at each timestep, it cannot guarantee evolutionary consistency with the previous step. Solutions with high modularity but low NMI may persist as non-dominated during evolution. Consequently, the algorithm fails to ensure that new solutions improve NMI.

This paper [108] proposes a novel multi-objective Bat Algorithm (BA) for community detection in dynamic social networks. The approach uses the Mean Shift algorithm to generate a high-quality initial population and redefines BA parameters (velocity, frequency, loudness, pulse rate) for this problem. It simultaneously optimizes modularity density3.21 and normalized mutual information 3.8 using a weighted objective approach, avoiding non-dominated sorting and crowding distance. A new mutation operator based on Mean Shift principles replaces random processes for solution generation.

Zeng et al.[180] proposed the concept of consensus communities to address DYNMOGA's limitation, introducing an evolutionary clustering algorithm called CCPSO. This approach extracts intrapopulation consensus communities from the non-dominated solutions of the previous timestep to preserve temporal knowledge. During the current timestep's evolution, the population 'votes' on these inherited consensus communities. Interpopulation consensus communities are then identified as the subset of consensus communities receiving support above a predetermined threshold, representing robust agreement between consecutive timesteps.

In [56], Gao et al. introduced a robust framework for dynamic community detection using decomposition-based multi-objective discrete particle swarm optimization (DYN-MODPSO). DYN-MODPSO simultaneously optimizes snapshot quality (commu-

nity structure precision) and temporal continuity. The framework employs modularity density (Q)3.21 to evaluate community structure quality at each timestep and normalized mutual information (NMI)3.11 to measure structural similarity between consecutive timesteps. Despite achieving favorable results on some networks, the algorithm suffers from premature convergence and low particle diversity.

Yin et al. [175] proposed Multi-results Discrete Particle Swarm Optimization called MRDPSO, an efficient multi-objective method enhancing the evolutionary clustering framework and particle swarm optimization (PSO). The key contributions include: (1) A recent-future reference strategy to refine initial clustering results, improving dynamic community detection accuracy; (2) Integration of a modified PSO into the evolutionary framework using this strategy; (3) A de-redundant random walk initialization method to generate diverse, high-quality solutions; (4) Multi-individual crossover and enhanced interference operators to avoid local optima.

Wang et al. [167] introduced a semi-supervised evolutionary autoencoder (sE-Autoencoder) for dynamic community detection. The method constructs a temporal matrix through nonlinear modeling, integrating autoencoder representations with evolutionary clustering. This framework simultaneously: (i) Optimizes community structures (ii) Preserves temporal consistency using vertex similarity. A key limitation is its dependence on prior knowledge of the initial community count, restricting applicability to networks with ambiguous or evolving community structures.

In another study, Besharatnia et al[15]. developed a method to enhance multi-objective optimization for dynamic community detection by integrating the Grey Wolf Optimizer (GWO) with the Label Propagation Algorithm (LPA). This hybrid approach, designated IGWO-LP, identifies non-overlapping community structures in dynamic networks. The method improves detection accuracy through integration with an evolutionary clustering framework

Abbood et al[1]. proposed Hidden Markov Model (HMM-MODCD) a novel framework for dynamic community detection that decomposes the problem into three components: intra-community structure, inter-community structure, and community evolution. The approach formulates these as a multi-objective optimization problem while modeling community evolution through Hidden Markov Models (HMMs) to identify probable temporal sequences. By integrating a multi-objective evolutionary algorithm with the Viterbi algorithm, HMM-MODCD simultaneously optimizes structural properties and maintains temporal coherence in dynamic networks.

Yu et al.[176] introduced cSWO a discrete Spider Wasp Optimization method enhanced with consensus community strategies for dynamic community detection. It restructures SWO's initialization, representation, and population updates while adapting its four core behaviors (search, following-escaping, nesting, mating) to discrete spaces.

Crucially, consensus communities serve as temporal knowledge during updates, guiding solution optimization through evolutionary transitions.

Dakiche et al.[40] introduced TCK-MBSO (Transferring Cliques and Quasi-cliques Knowledge with Multi-Objective Bee Swarm Optimization), a novel dynamic community detection method. Its core innovation lies in transferring critical clique and quasi-clique structures from historical communities to the current detection phase. This knowledge transfer enhances both detection accuracy and temporal consistency. Embedded within a multi-objective bee swarm optimization framework, TCK-MBSO significantly advances dynamic community identification capabilities.

Within evolutionary clustering frameworks for temporal networks, Ma and Dong [104] pioneered two Evolutionary Non-negative Matrix Factorization (ENMF) approaches for dynamic community detection. They established fundamental theoretical equivalences: ENMF ≡ evolutionary modularity density optimization and ENMF ≡ evolutionary spectral clustering, providing a foundation for hybrid algorithms. Building on this, they developed semi-supervised ENMF (sE-NMF) which uniquely integrates prior knowledge directly into the ENMF objective functiondistinguishing it from conventional semi-supervised methods. Crucially, sE-NMF avoids local optima without increasing time complexity, offering computational efficiency alongside enhanced optimization capability.

Jiao et al.[75] developed a probabilistic NMF framework for dynamic community detection, simultaneously modeling temporal/static community structures and node significance. The method identifies three core matrices (community membership, snapshot similarity, inter-timestep transitions), formulates detection via network generation principles, and solves optimization through gradient descent.

Yu et al.[177] introduced ECGNMF, an algorithm using Non-negative Matrix Factorization to dynamically update communities across temporal snapshots. It uniquely integrates both community-level (mesoscopic) and node-level (microscopic) evolutionary information into its cost function, with regularization ensuring consistency by limiting node transitions between timesteps.

Li et al.[93] developed NE2NMF, integrating Network Embedding and Evolutionary NMF for dynamic community detection. The algorithm employs third-order temporal smoothness (past/current/future snapshots) to track community evolution and reduces time complexity by decomposing the network embedding matrix leveraging the established equivalence between ENMF and network embedding.

To address the limitations of shallow NMF in capturing complex network structures, Zhang et al.[182] proposed Constrained Symmetric Deep NMF (CSDNMF), integrating deep autoencoders with NMF. The model delivers three innovations: (1) Hierarchical feature learning: Deep autoencoders extract latent features linking input networks to community assignments. (2) Topology preservation: Graph regularization maintains

network structure.(3) Undirected network optimization: Symmetry regularization enforces representation parity. Evaluations across eight real-world networks demonstrate CSDNMF's superiority over state-of-the-art community detection methods.

Kojaku et al[83].demonstrated that shallow linear graph neural networks (e.g., node2vec) can achieve the information-theoretic detectability limit for communities. Building on this foundation, liu et al [98] study introduced a novel framework integrating community detection algorithms with diverse Graph Neural Network (GNN) models to enhance link prediction in scientific literature networks. The integration of the Louvain algorithm with GNNs consistently improved performance across all tested architectures. Notably, Louvain integration with the GAT model demonstrated significant performance gains. Similar improvements occurred when combining Louvain with other GNNs, validating the robustness and efficacy of incorporating community-level features.

Table 3.2: Comprehensive Analysis of Community Detection Methods

Method	Category	Complexity	Topology Suitability	Size Class	Key Characteristics
Static Approaches: Heuristic Methods					
Newman (2004) [114]	Greedy	$O((m+n)n)$	Hierarchical	BSRP	Foundational modularity optimization Quadratic complexity
Clauset et al. [38]	Greedy	$O(md\log n)$	Scale-free	BMRP	Heap-based acceleration Resolution limit ($\sqrt{2m}$)
Louvain [16]	Greedy	$O(m)$	Large-scale	BLRP	Fast two-phase aggregation Disconnected communities
Leiden [163]	Greedy	$O(m)$	Disconnected	BLRP	Connectivity guarantee 25% quality improvement over Louvain
LPAm+ [100]	Label Prop.	$O(n\log n)$	Homogeneous	BSRP	Linear scalability Random walk dependency
GenLouvain [112]	Multiplex	$O(Lm)$	Multilayer	BMRP	Cross-layer dependency handling Modularity-bound
Static Approaches: Metaheuristic Methods					
GA-Net [128]	Evolutionary	$O(n^2)$	Small networks	BSRP	Locus-based encoding Premature convergence
LMOEA [37]	Evolutionary	$O(n^2)$	Overlapping	BMRP	Conflicting objective handling Parameter sensitivity
DPSO [30]	Swarm	$O(n^2)$	Signed	BSRP	Negative edge handling Limited scalability
MSIG [147]	Iterative	$O(n^2)$	Constructive	BSRP	Greedy Constructive Procedure Limited intensification
SFCSA [25]	Hybrid	$O(knd)$	Attributed	BLRP	Node2vec + cuckoo search Feature dependency
CD-GNN [98]	Emerging	$O(knd)$	Integrated	BLRP	Community-enhanced link prediction Framework complexity
Incremental Approaches: Local Updates					
DynaMo [185]	Localized	$O(\log k)$	Slow-evolving	BMRP	Handles 6 update types Global drift over time
Bouhatem [24]	Localized	$O(\Delta n)$	Expanding	BLRP	Density optimization Fails during community fission
Δ-screening [179]	Localized	$O(\Delta\log n)$	Batch updates	BLRP	Selective node reevaluation Blind to radical changes
FCDDCN [9]	Global	$O(m\log n)$	Real-time	BMRP	Adjacency list updates CNM algorithm dependency
Incremental Approaches: Multi-objective					
DYNMOGA [53]	EMOO	$O(gn^2)$	Multiobjective	BLRP	NMI-modularity tradeoff Temporal inconsistencies
CCPSO [180]	EMOO	$O(gn^2)$	Consensus	BLRP	Temporal knowledge preservation Threshold sensitivity
MRDPSO [175]	EMOO	$O(gn^2)$	Enhanced	BLRP	De-redundant initialization Fails on community expansion
TCK-MBSO [40]	EMOO	$O(gn^2)$	Clique-aware	BLRP	Clique structure transfer Dependency on clique existence
Incremental Approaches: Embedding Methods					
NE2NMF [93]	Matrix	$O(knd)$	Embedded	BLRP	3rd-order temporal smoothness Embedding drift risk
CSDNMF [182]	Matrix	$O(knd)$	Deep	BLRP	Graph regularization Undirected networks only
ECGNMF [177]	Matrix	$O(n^2)$	Multi-scale	BMRP	Micro-macro integration Regularization sensitivity
sE-Autoencoder [167]	Hybrid	$O(n^2)$	Semi-supervised	BLRP	Nonlinear temporal modeling Requires initial community count

Key:
Size Class: BSRP (Small <1k nodes), BMRP (Medium 1k-100k), BLRP (Large >100k)
Complexity: n=nodes, m=edges, k=communities, L=layers, g=generations, Δ=batch size

3.7 Conclusion

This chapter outlines definitions of the community discovery problem across simple, multiplex, and dynamic networks. It also reviews evaluation metrics for quantifying detection quality, noting their role as potential fitness functions in optimization. The chapter culminates in a call for a paradigm shift, proposing that future advancements require a transition from traditional methods toward Next-Generation Hybrid Metaheuristics for Community Detection (HMCD). Central to this new paradigm is the integration of network embedding techniques within HMCD frameworks.

CHAPTER 4

AN ENHANCED BAT ALGORITHM USING CLUSTERING COEFFICIENT FOR COMMUNITY DETECTION IN COMPLEX NETWORKS

Bio-inspired algorithms turn distributed, simple behaviors into powerful problem-solving systems.

4.1 Introduction

This chapter presents the Clustering Coefficient Discrete Bat Algorithm (CC-DBAT), a new method for detecting communities in non-overlapping networks. CC-DBAT enhances the standard DBAT by using the clustering coefficient to initialize solutions and employs fast local move and random neighbor techniques to improve optimization. Tests on synthetic and real-world data show it outperforms other bat algorithms and converges faster.

4.2 Bat algorithm

The Bat Algorithm (BAT) is a meta-heuristic optimization technique inspired by the sophisticated echolocation capabilities of microbats, developed by Yang[174]. The algorithm models this biological phenomenon using a set of heuristic rules, as outlined in the pseudocode provided in Algorithm 1:

- Echolocation enables distance perception and discrimination between prey and barriers.

- Search is conducted through random flight with velocity V_i at position Pos_i, utilizing adjustable frequency (or wavelength) and a pulse emission rate $Pr \in [0,1]$ that is contingent on the target's proximity.
- The intensification of the search is controlled by increasing the pulse emission rate and decreasing the loudness lds as the bat converges on its prey.

The algorithm begins by initializing a population of bats, each with a defined position (Pos_i), velocity (V_i), frequency ($\mathscr{F}i$), loudness (Lds), and pulse emission rate (Pr). These parameters are then iteratively refined according to a set of update equations until a predetermined maximum iteration count ($Itermax$) is satisfied.

$$\mathscr{F}_i = F_{min} + (F_{min} - F_{max})\beta \tag{4.1}$$

$$Ve_i^t = Ve_i^{t-1} + (Pos_{best} - Pos_i^{t-1})\mathscr{F}_i \tag{4.2}$$

$$Pos_i^t = Pos^{t-1} + Ve_i^t \tag{4.3}$$

Equation 4.1 uses a random value β (between 0 and 1) and the frequency bounds f_{min} and f_{max} to calculate a new frequency. This frequency updates the velocity (Ve_i^t) and position (Pos_i^t) for each dimension of the i^{th} bat, guiding it toward the global best solution (Pos_{best}). Initially, the loudness (lds_i^0) and pulse emission rate (Pr_i^0) are randomly set within the ranges [1, 2] and [0, 1], respectively.

$$Pos_{new} = Pos_{old} + loud^t \tag{4.4}$$

$$lds_i^{t+1} = \alpha \times lds_i^t \tag{4.5}$$

$$Pr_i^{t+1} = Pr_i^0[1 - exp^{-\lambda t}] \tag{4.6}$$

$$\tag{4.7}$$

4.2.1 Binary bat algorithm

While the standard Bat Algorithm (BAT) is designed for continuous-valued problems, Nakamura et al.[113] extended it with Xin-She Yang to create the Binary Bat Algorithm (BBA) for discrete, binary problems. In the BBA, the search space is modeled as an n-dimensional hypercube. Each bat's position is a Boolean array (a string of 0s and 1s) where each bit indicates the exclusion or inclusion of a specific attribute. The position is updated and evaluated using a sigmoid function (Eq. (4.8)), as defined in Eq (4.9).

$$S(v_j^i) = \frac{1}{1 + exp^{-v_j^i}}, \tag{4.8}$$

$$\begin{cases} 1 & \text{if } S(v_j^i) > \rho \\ 0 & \text{Otherwise} \end{cases} \tag{4.9}$$

Algorithm 1 BAT algorithm

Generate the initial population of bats Pos.
Initialize velocity V_i for each bat.
Initilize loudness lds_0, maximum number of iterations $Iter_{Max}$, and pulse rate Pr_0 .
repeat
  for all bat in Population **do**
    Update the velocity vector V_i according to equation (4.2)
    Update the position according to equation (4.3)
    Generate a random number r_1 between 0 and 1.
    if ($Pr_i < r_1$) **then**
      Generate a local solution around one of the selected best solutions;
    end if
    Generate a random number r_2 between 0 and 1.
    if (r2 < lds_i) and (fitness(Pos_i) < fitness(Pos_{best}) **then**
      Accept the new solutions
    end if
    Increase Pr_i and reduce lds_i (4.5)(4.6)
  end for
  Update the Best position Pos^{best}
until (Maximum number of iterations is reached)
Return Pos^{best}

4.3 The proposed algorithm

The CC-DBAT algorithm is specifically proposed to detect community structures in complex networks by optimizing the modularity function. This approach is rooted in the bat algorithm framework [174].

The CC-DBAT algorithm builds upon the concepts employed in the Discrete BAT-Modified (DBAT-M) Optimization Algorithm [2]. However, it enhances the initial population generation by using the clustering coefficient metric [170], with each bat's position represented through locus-adjacency representation [144]. These representations are then decoded into valid partitions. The CC-DBAT algorithm iteratively updates the position and velocity of each bat, employing the fast local move technique [122] and a random neighbor community method [162] to refine solutions until no further improvements in modularity can be achieved. A high-level overview of the CC-DBAT algorithm

is presented in Algorithm 2, which is based on the foundational principles of DBAT-M [2].

4.3.1 Problem definition

A network of community is modeled as a simple undirected graph $G(V,E)$ 2.2.1. The community detection problem with respect to G asks for a partition $\mathscr{C} = \{C_1, C_2, \cdots, C_k\}$ such that $\bigcup_{r=1}^{k} C_r = V$ and all subsets of $\mathscr{C}$ are pairwise disjoint, that is, for all $1 \leq i < j \leq k$ it holds that $C_i \cap C_j = \emptyset$.

The commonly used metric in the context of community detection for evaluating the quality of a given partition $\mathscr{C}$ is the modularity function [116], is formally define in 3.15.

The objective of community detection is therefore to find a partition of the vertex set Valso referred to as a graph clusteringinto disjoint communities that maximizes the modularity measure 3.15.

Algorithm 2 CC-DBAT algorithm

Input: Graph $G(V,E)$, loudness A, and pulse rate R
Output: X^{best}

$\mathscr{X} \leftarrow$ Generate the initial population of solutions X.
Initialize velocity vector $\mathscr{V}_i$ for each bat.
$X^{best} \leftarrow$ **argmax** $\{Q(X_i) | X_i \in \mathscr{X}\}$
repeat
  for all X_i in $\mathscr{X}$ **do**
    Update the velocity vector $\mathscr{V}_i$ according to equation (4.11)
    Update the position vector X_i according to equation (4.13)
    Update a random number r_1 between 0 and 1.
    if $(R < r_1)$ **then**
      $X_i \leftarrow$ FastLocalMove(X_i) ▷ See algorithm 4
      if $(Q(X_i) > Q(X^{best}))$ **then** $X^{best} \leftarrow Q(X_i)$
      end if
    end if
    $X_* \leftarrow$ RandomNeighborCommunity(X_i)
    Generate a random number r_2 between 0 and 1.
    if $(A > r_2$ **and** $Q(X^*) > Q(X_i))$ **then**
      $X_i \leftarrow X_*$
    end if
    Increase R and reduce A
  end for
  Update the Best position X^{best}
until (Maximum number of iterations is reached)
Return X^{best}

4.3.2 Generate initial population

An initial bat position is encoded using locus-adjacency representation, with each position corresponding to a node in the graph. In this setup, each node v connects to a neighboring node v, indicating they belong to the same community. Instead of linking to a random neighbor, we use the clustering coefficient (CC) to connect each node to its neighbor with the highest CC. The CC measures a node's connectivity among its neighbors; the more interconnected they are, the more likely they belong to the same community [144]. The clustering coefficient of node v is defined as follows:

$$CC_v = 2 \times \frac{L_v}{d_v(d_v - 1)} \tag{4.10}$$

Here, d_v denotes the degree of node v, and L_v is the total number of links among v's neighbors, excluding itself. The clustering coefficient CC_v ranges from 0 to 1, where $CC_v = 0$ indicates no links among neighbors and $CC_v = 1$ signifies that all neighbors are interconnected.

The initialization process for each bat position includes:

- Randomly select a node v.
- Select the neighbor u with the highest CC.
- If multiple neighbors share the highest CC, select one randomly.
- If $d_v = 1$, associate v with u.
- If $d_v = 0$, assign v to a singleton community.

We use the genotype to refer to the locus-based adjacency representation. After initializing the bat positions, a decoding process [150] is conducted to obtain a valid partition, as shown in Fig. 4.1. The process of decoding is outlined in Algorithm 3.

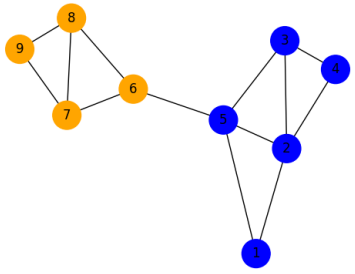
4.3.3 Update velocity and position

In the original algorithm, velocity is defined as the difference between the current individual's position and the position of the current best individual. However, in a discrete encoding scheme, this difference cannot be computed directly through subtraction. To address this, a new velocity formula based on the XOR operation between the current individual and the current best individual has been introduced in [156]. Conceptually, each bat updates its velocity by learning from the current best solution. This learning process essentially involves comparing the two positions. The **XOR** operation effectively captures the discrepancy between two network partitionssince each bat represents one such partition. For example:

Algorithm 3 Decode of genotype

Input: Genotype[n]
Output: X

$current\_community \leftarrow 1$
for each i in 1 to n **do**
    $X_i \leftarrow -1$
end for
for each i in 1 to n **do**
    $Label \leftarrow 1$
    if $X_i = -1$ **then**
        $X_i \leftarrow current\_community$
        $neighbor \leftarrow Genotype[i]$
        $previous_{Label} \leftarrow i$
        $label \leftarrow label + 1$
        while $X_{neighbor} = -1$ **do**
            $previous_{label} \leftarrow neighbor$
            $X_{neighbor} \leftarrow current\_community$
            $neighbor \leftarrow Genotype_{neighbor}$
            $label \leftarrow label + 1$
        end while
        if $X_{neighbor} \neq current\_community$ **then**
            $label \leftarrow label - 1$
            while $label \geq 1$ **do**
                $X_{previous_{label}} \leftarrow X_{neighbor}$
                $label \leftarrow label - 1$
            end while
        else
            $current\_community \leftarrow current\_community + 1$
        end if
    end if
end for
$number\_of\_community \leftarrow current\_community$
Return X



NODE	1	2	3	4	5	6	7	8	9
CC	1	0.5	0.66	1.0	0.3	0.3	0.6	0.6	1
Locus Based adjacency	2	4	4	3	1	7	8	9	7

decode

1	2	3	4	5	6	7	8	9
1	1	1	1	1	2	2	2	2

Figure 4.1: Locus-based adjacency representation and the correspondent solution after decoded the solution

- If $X_i = X_i^*$ (where X_i^* is the current best individual) then $X_i \oplus X_i^* = 0$
- If $X_i \neq X_i^*$, then $X_i \oplus X_i^* = 1$.

The position and velocity of each individual is described as follow :

- The velocity $\mathcal{V}$ is formally defined as follow [156]:

$$\mathcal{V}_i^t = \mathcal{V}_i^{t-1} + (X_i^t \oplus X_i^*) f_i \tag{4.11}$$

$$\mathcal{V}_i^t = \begin{cases} 1 & \text{if } rand() < Sig(\mathcal{V}_i^t) \\ 0 & \text{if } rand() \geq Sig(\mathcal{V}_i^t) \end{cases} \tag{4.12}$$

- For update the positions:

$$X_i^t = \begin{cases} X_i^t & \text{if } \mathcal{V}_{id}^t = 0 \\ X_i^t & \text{if } \mathcal{V}_{id}^t = 1, rand() \geq r_i \\ X_t^{new} & \text{if } \mathcal{V}_{id}^t = 1, rand() < r_i \end{cases} \tag{4.13}$$

4.3.4 Fast Local Move Procedure

We enhance CC-DBAT by integrating the Fast Local Move (FLM) heuristic, a core component of the Louvain Prune Algorithm [122], as a replacement for the conventional local search (LS) procedure. This heuristic has been demonstrated to reduce computational time by up to 90 % compared to the original method while maintaining solution quality. FLM strengthens the algorithm's intensification capability, having been successfully employed as a reconstruction phase in an iterated greedy algorithm for community detection [160].

Algorithm 4 outlines the steps of this process. Initially, a set L is initialized with all nodes from V. The algorithm then iterates over the elements of L in random order until the set is empty and no further modularity improvement is possible. In each iteration, a node v is selected uniformly at random from L. The modularity gain ΔQ is computed for moving v to every neighboring community that contains at least one adjacent node. The community C^{best} that yields the highest gain is selected as the new community for v (lines 710). Subsequently, all neighbors of v that do not belong to C^{best} and are not already in L are added to L (lines 1415).

The modularity gain $\Delta Q(v,C)$ resulting from assigning node v to community C is defined as follows [16, 122]:

$$\Delta Q(v,C) = |N_v \cap C| - \frac{d_v \cdot \sum_{u \in C} d_u}{2m^2} \tag{4.14}$$

where $|N_v \cap C|$ represents the number of neighbors of node v that belong to community C, and $\sum_{u \in C} d_u$ is the sum of degrees of all nodes in C.

Algorithm 4 Fast_Local_Move procedure.

Input: S_0, L
Output: S

1: **while** (L is not empty) **do**
2: Select a node v uniformly at random from L.
3: $L \leftarrow L \setminus \{v\}$
4: $S_v \leftarrow \{C | C \in S_0, N_v \cap C \neq \emptyset\}$ ▷ N_v represents the open neighborhood of vertex v
5: $\Delta Q^{best} \leftarrow$ **max** $\{\Delta Q(v,C) | C \in S_v\}$
6: **if** ($\Delta Q^{best} > 0$) **then**
7: $C^{best} \leftarrow$ **argmax** $\{\Delta Q(v,C) | C \in S_v\}$
8: Let C_v represents the community associated with node v.
9: $C^{best} \leftarrow C^{best} \cup \{v\}$
10: $C_v \leftarrow C_v \setminus \{v\}$
11: **if** (C_v is empty) **then**
12: $S_0 \leftarrow S_0 \setminus \{C_v\}$
13: **end if**
14: $P \leftarrow N_v \setminus C^{best}$
15: $L \leftarrow L \cup (P \setminus L)$
16: **end if**
17: **end while**
18: $S \leftarrow S_0$
19: **Return** $S = \{C_1, C_2, C_3, \ldots, C_k\}$

4.3.5 Random neighbor community

Two key search mechanisms intensification and diversification are effectively balanced in the CC-DBAT algorithm, enhancing its ability to achieve high-quality partitions. This

phase generates random solutions by relocating nodes into neighboring communities selected at random [162]. According to the authors, choosing a random neighbor for the Louvain algorithm serves several purposes. This modification reduces the method's greediness, increasing its ability to explore the solution space. The strategy is efficient as it leverages the network structure: since a node is likely to belong to a community containing many of its neighbors, randomly selecting a neighbor's community is probably a good choice. Formally, the selection probability for any community is proportional to the number of connections the node has within it ($L(i,c)/d_i$). This ensures that "better" communities (those with more connections) are sampled more frequently, allowing the algorithm to concentrate on the most promising options without the computational cost of evaluating every single one.

Algorithm 5 Select the random neighbor community.

function RANDOM NEIGHBOR COMMUNITY(Node i)
  return random $\sigma \in \{\sigma_i \mid (ij) \in E(G)\}$
end function

4.4 Experimental Evaluation

This section highlights the outcomes produced by the CC-DBAT algorithm, along with a detailed analysis of the computational experiments conducted to evaluate its performance against two established algorithms within the same metaheuristic framework: the DBAT-M algorithm [2] and the DBAT algorithm [156], both of which are based on the bat algorithm. The analysis incorporates both real and synthetic networks to provide a comprehensive assessment.For the real networks, the performance metrics for the DBAT-M and DBAT algorithms are sourced from their original publications, ensuring a valid comparison. In contrast, for the synthetic networks, where no existing results were available, we undertook the task of reimplementing these two metaheuristics. This allowed us to rigorously evaluate their performance across a variety of generated instances. All algorithms were implemented in Python with the NetworkX package. The experiments were run on an Intel Core i5-13600KF CPU (5.1 GHz) with 16 GB RAM, using the Ubuntu 22.04 LTS operating system.

4.4.1 Problem instances

The evaluation set comprises both real-world networks and synthetic networks, the latter generated based on specific parameters.

4.4.1.1 Real-world networks

The first category comprises eight real networks that have been widely tested across various algorithms. These networks vary in size, classified from smallest to largest as follows: the Train Bombing Network (22 nodes) [88], the Windsurfers Network (30 nodes)[88], Zachary's Karate Club (34 nodes)[178], the Taro Exchange (42 nodes)[88], the Dolphins Social Network (62 nodes)[103], the Co-purchased Political Books network (polbooks) (105 nodes)[115], the American College Football Network (115 nodes)[59], and the Jazz Musicians Network (198 nodes)[60].

4.4.1.2 Synthetic Networks

The performance of the CC-DBAT algorithm was further assessed using established synthetic benchmark networks: the Girvan-Newman (GN) [116] and LFR [90] benchmarks1. Two sets of benchmarks were generated, each consisting of seven distinct networks. The parameter configurations for these networks are summarized in Table 4.1.

Table 4.1: Parameter settings of GN and LFR benchmark networks

Network	LFR	GN
N	1000	128
K_{avg}	15	16
K_{max}	50	16
$MinC$	10	32
$MaxC$	50	32
μ	0.05 - 0.7	0.1 - 0.5
θ_1	2	2
θ_2	1	1

4.4.2 Evaluation criteria

Two standard metrics were used to evaluate the effectiveness of the proposed approach: modularity (Q) 3.15 and Normalized Mutual Information (NMI)3.8.

4.4.3 Results and Discussion

4.4.3.1 Convergence analysis

The increase in modularity values for all algorithms was examined to analyze the convergence behavior of the CC-DBAT algorithm in comparison to other bat-based metaheuristics. For this analysis, the polbooks network was selected, as shown in Fig. 4.2. The x-axis indicates the number of iterations performed by the algorithm, while

Table 4.2: Experimental results of CC-DBAT and the compared algorithms on real networks

Networks	Criterion	DBAT	DBAT-M	CC-DBAT
Karate	Q_{max}	0.4043	0.4020	**0.4198**
	Q_{avg}	0.4020	0.4020	**0.4198**
	Q_{std}	0.0003	0.0	**0.0**
	NMI	**0.6994**	0.6873	0.6872
	$Time(s)$	4.18	3.152	**2.9**
Dolphins	Q_{max}	**0.5277**	0.5277	0.5161
	Q_{avg}	**0.5266**	0.5257	0.5153
	Q_{std}	0.0005	0.001	**0.0003**
	NMI	0.5792	0.6454	0.4316
	$Time(s)$	9.5	7.11	**5.8**
Football	Q_{max}	0.5937	**0.600**	0.5763
	Q_{avg}	**0.5917**	0.5857	0.5695
	Q_{std}	0.006	0.005	**0.004**
	NMI	0.8292	0.8722	0.7432
	$Time(s)$	25.98	18,05	**15.07**
Polbooks	Q_{max}	0.4794	0.5264	**0.5264**
	Q_{avg}	0.4631	0.5264	**0.5264**
	Q_{std}	0.02	**0.00**	**0.00**
	NMI	0.4800	0.5956	**0.5956**
	$Time(s)$	17.87	12.59	**12.10**
Taro exchange	Q_{max}	0.4241	0.4532	**0.4543**
	Q_{avg}	0.3871	0.3982	**0.4543**
	Q_{std}	0.02	0.04	**0.00**
	$Time(s)$	5.7	2.97	**2.53**
Windsurfers	Q_{max}	0.2600	0.2779	**0.3130**
	Q_{avg}	0.2433	0.2650	**0.3099**
	Q_{std}	0.01	0.017	**0.0014**
	$Time(s)$	10.7	5.6	**5.4**
Train-bombing	Q_{max}	0.4291	0.4315	**0.4323**
	Q_{avg}	0.3571	0.4117	**0.4318**
	Q_{std}	0.07	0.01	**0.0003**
	$Time(s)$	15.43	**6.9**	7.04
Jazz	Q_{max}	0.3891	0.4431	**0.4447**
	Q_{avg}	0.3421	0.4414	**0.4447**
	Q_{std}	0.021	0.004	**0.0**
	$Time(s)$	50.86	36.75	**35.89**

the y-axis represents the corresponding modularity values, reflecting the quality of the detected community structures at each iteration. The illustration indicates that the CC-DBAT technique reaches the global optimum within just a few iterations.

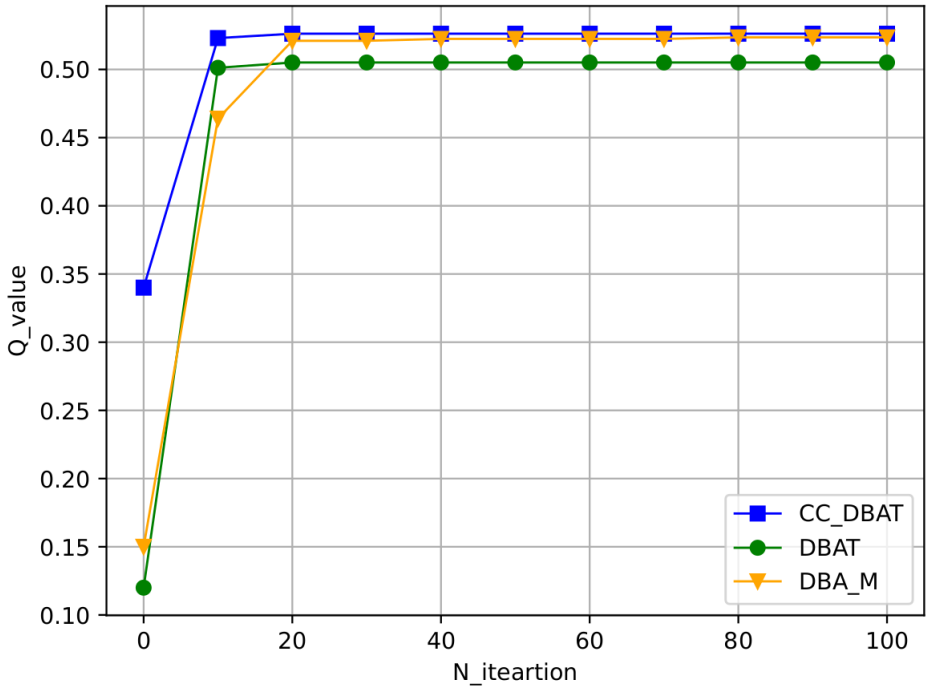


Figure 4.2: The convergence analysis of the CC-DBAT, DBAT, and DBAT-M algorithms on the Polbooks network.

4.4.3.2 Performance on real-world networks

The proposed CC-DBAT algorithm was evaluated by comparing its performance against other bat-based methods on a range of real-world networks. Each method was executed independently 10 times for every instance. Table 4.2 presents the results of various bat-based algorithms, detailing the maximum modularity (Q_{max}), the average modularity (Q_{avg}), and the standard deviation of modularity values (Q_{std}) to summarize their performance.

Additionally, it provides the maximum NMI values for four networks with known community structures. As shown in Table 4.2, the CC-DBAT achieved competitive results on all networks except for the Football and Dolphins networks. Notably, the CC-DBAT consistently reached the best modularity values with high stability compared to the other metaheuristics, DBAT and DBAT-M. Furthermore, the CC-DBAT effectively detected high-quality community structures in real networks with established ground truths.

4.4.3.3 Performance evaluation on synthetic networks

The synthetic networks, having known community structures, allow us to use NMI as the evaluation criterion. It is important to note that as μ increases, defining a valid partition becomes more challenging. We executed all the compared algorithms with ten

independent runs on each GN and LFR network. As shown in Fig. 4.3, the CC-DBAT algorithm successfully reveals valid partitions in all GN networks where $\mu < 0.45$, as the NMI value equals one. The CC-DBAT also detects high-quality partitions when $\mu = 0.5$. The results for the LFR networks, as depicted in Fig. 4.4, show that all BAT-based meta-heuristics can accurately define the valid community structure of networks when $\mu < 0.35$. Our technique can reveal a good partition even as μ increases. Therefore, the CC-DBAT significantly outperforms the DBAT-based methods on synthetic networks.

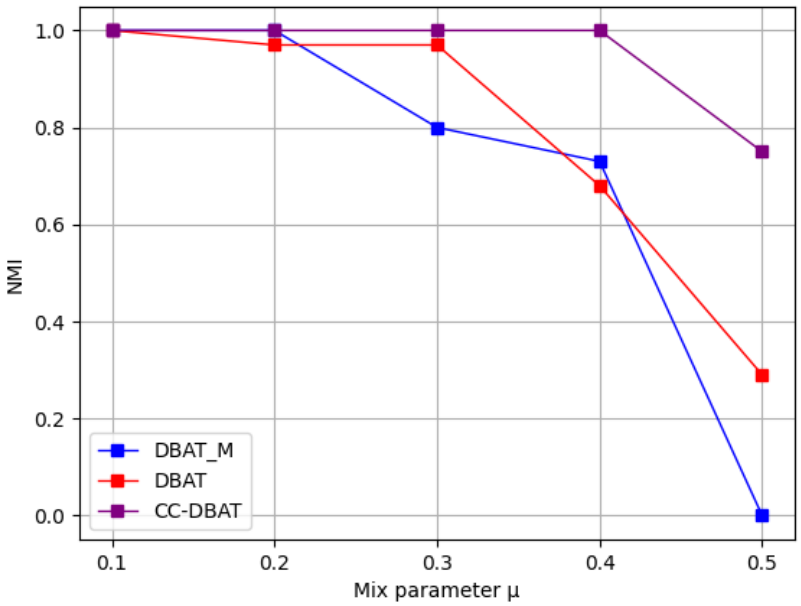


Figure 4.3: The maximum NMI achieved by the DBAT-M, CC-DBAT, and DBAT algorithms on GN networks.

4.5 Conclusion

This chapter introduces the Clustering Coefficient Bat Algorithm (CC-DBAT), a novel hybrid metaheuristic for detecting communities in complex networks. CC-DBAT initializes the population using node clustering coefficients and refines it through a fast local move heuristic and a random neighbor strategy, fostering solution diversity and rapid convergence. Evaluations on both synthetic and real-world networks confirm its competitive performance against established bat algorithm variants for community detection, focusing specifically on non-overlapping communities.

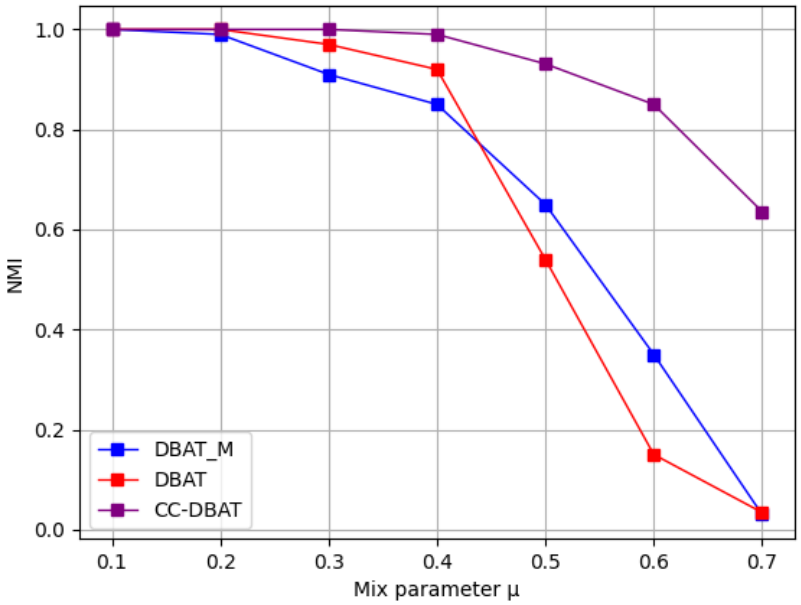


Figure 4.4: The maximum NMI achieved by DBAT-M, CC-DBAT, DBAT algorithms on LFR networks.

CHAPTER 5

COMPLEX NETWORK COMMUNITY DISCOVERY USING FAST LOCAL MOVE ITERATED GREEDY ALGORITHM

An algorithm must be seen to be believed.Donald Knuth

5.1 Introduction

This chapter presents the Fast Local Move Iterated Greedy (FLMIG) algorithm for detecting communities in complex networks. FLMIG is an enhancement of the Iterated Greedy framework, augmented with the Louvain Prune heuristic to maximize modularity for non-overlapping communities. The algorithm's strength lies in its fusion of the Fast Local Move technique's efficient local search with a robust iterative refinement process involving destruction and reconstruction. The chapter's structure includes a discussion of the iterative greedy framework, a detailed outline of the FLMIG design, a computational complexity analysis, and extensive experimental validation. These experiments benchmark FLMIG against leading methods like Leiden, Iterated Carousel Greedy, and Louvain Prune, demonstrating its scalability, parameter efficiency, and strong performance on large-scale networks.

5.2 Iterated Greedy

5.2.1 Greedy Construction Heuristics

Constructive algorithms build solutions incrementally from an empty starting point. These methods iteratively extend a partial solution by adding new components until a complete solution is formed. The selection of the next component to add is guided by a heuristic function that estimates the benefit of each candidate addition. A basic yet widely used variant is the greedy algorithm, which consistently selects the component with the highest heuristic value at each step, as described in Algorithm6.

Greedy construction heuristics are widely employed in combinatorial optimization for several reasons. They are computationally efficient and yield solutions substantially better than those generated randomly or by simple stochastic methods. Moreover, they serve as highly effective initializers for more advanced metaheuristicssuch as iterative improvement, tabu search, simulated annealing, and memetic algorithmsand provide the foundational structure for sophisticated approaches like GRASP[51], Ant Colony Optimization (ACO)[46], and squeaky wheel optimization[76].

Algorithm 6 Greedy_Constructive_Heuristic

1: $S \leftarrow \emptyset$
2: **while** S is an incomplete solution **do**
3: Select the highest-rated solution component, C_p.
4: $S \leftarrow S + C_p$
5: **end while**

5.2.2 Iterated Greedy Framework

The core principle of the Iterated Greedy (IG) algorithm is to iteratively apply a greedy construction process. It begins by generating an initial complete solution and then enters a main loop composed of two phases: destruction and construction. In the destruction phase, a portion of the current solution is removed, resulting in a partial solution. The subsequent construction phase uses a greedy heuristic to rebuild a new complete solution from this partial one. Finally, an acceptance criterion determines whether the new solution or the original one is used for the next iteration. Renowned for its simplicity, IG is a stochastic iterative process that relies on few control parameters and operates effectively without extensive problem-specific customization, unlike more complex metaheuristics.

Originally proposed by Ruiz and Stützle[143] for permutation flow shop scheduling, the IG framework has since proven highly successful in tackling a broad spectrum of NP-hard combinatorial optimization problems [102]. This includes community detection in networks [84, 94, 147] and various other graph-based problems[21–23, 32].

As outlined in Algorithm 7, a typical IG algorithm starts by generating an initial solution. It then enters its main loop, which repeatedly executes the three key procedures: Destruction, Construction, and AcceptanceCriterion. It is important to note that the construction heuristics used for generating the initial solution and for the reconstruction phase within the loop can be distinct, potentially employing different greedy strategies.

Algorithm 7 The iterated greedy framework

1: $S \leftarrow$ Generate_Initial_Solution
2: **repeat**
3: $S^P \leftarrow$Destruction_Solution (S)
4: $S^* \leftarrow$ Reconstruction_Solution (S^P)
5: $S \leftarrow$ Acceptance_Criterion (S^*,S)
6: **until** (Termination condition is satisfied)
7: **Return** S

5.3 Proposed approach

We introduce the Fast Local Move Iterated Greedy (FLMIG) algorithm, an approach grounded in the Iterated Greedy (IG) framework [143], designed to address the community detection problem by maximizing the modularity metric. The formal definition of this problem is provided in the previous chapter 4.3.1. As outlined in Algorithm 8, the method begins by generating an initial solution via the Generate_Initial_Solution procedure, which employs the Fast Local Move (FLM) heuristic [122]. This initial solution then undergoes repeated cycles of destruction and reconstruction until a predefined termination condition is met.

During the destruction phase (line 4), the Destruction_Solution procedure partially dismantles the current solution by randomly removing certain nodes and reassigning each to its own singleton community. This intentional perturbation helps escape local optima and encourages exploration of new regions in the solution space.

The subsequent reconstruction phase (line 5) uses the Reconstruction_Heuristic to rebuild the solution, reintegrating singleton communities into more promising neighboring communities. FLMIG enhances this process by incorporating an advanced variant of the Louvain Prune algorithm [122] and introduces an additional refinement step to resolve locally disconnected communities, thereby accelerating convergence and improving efficiency.

Finally, the Select_Next_Solution procedure (line 6) decides whether to accept the newly reconstructed solution for the next iteration, completing one full cycle of the algorithm.

Algorithm 8 Fast local move iterated greedy algorithm

Input: Graph $G(V,E)$
Output: $S^{best} = \{C_1, C_2, C_3, \ldots, C_k\}$

1: $S \leftarrow$ Generate_Initial_Solution (G) ▷ See algorithm 9
2: $S^{best} \leftarrow S$
3: **repeat**
4: $(S^{dest}, L^{dest}) \leftarrow$Destruction_Solution ($S$) ▷ See algorithm 10
5: $S^* \leftarrow$ Reconstruction_Heuristic (S^{dest}, L^{dest}) ▷ See algorithm 11
6: $S \leftarrow$ Select_Next_Solution (S^*, S) ▷ See algorithm 12
7: **if** $(Q(S) > Q(S^{best}))$ **then**
8: $S^{best} \leftarrow S$
9: **end if**
10: **until** (Termination condition is satisfied)
11: **Return** S^{best}

Figure 5.1 presents a graphical example of the various components of the FLMIG algorithm as applied to the karate club network.

5.3.1 Generate_Initial_Solution procedure

Similar to other Iterated Greedy (IG) approaches, the FLMIG algorithm initiates by constructing an initial solution through the Generate_Initial_Solution procedure (Algorithm 9). This method initially assigns each node $i \in V$ to its own single-node community C_i, resulting in a trivial partition comprising n such communities. This preliminary solution is then refined using the Fast_Local_Move procedure (line 6), which efficiently generates a high-quality initial solution. A detailed explanation of this process is provided in Algorithm 4.

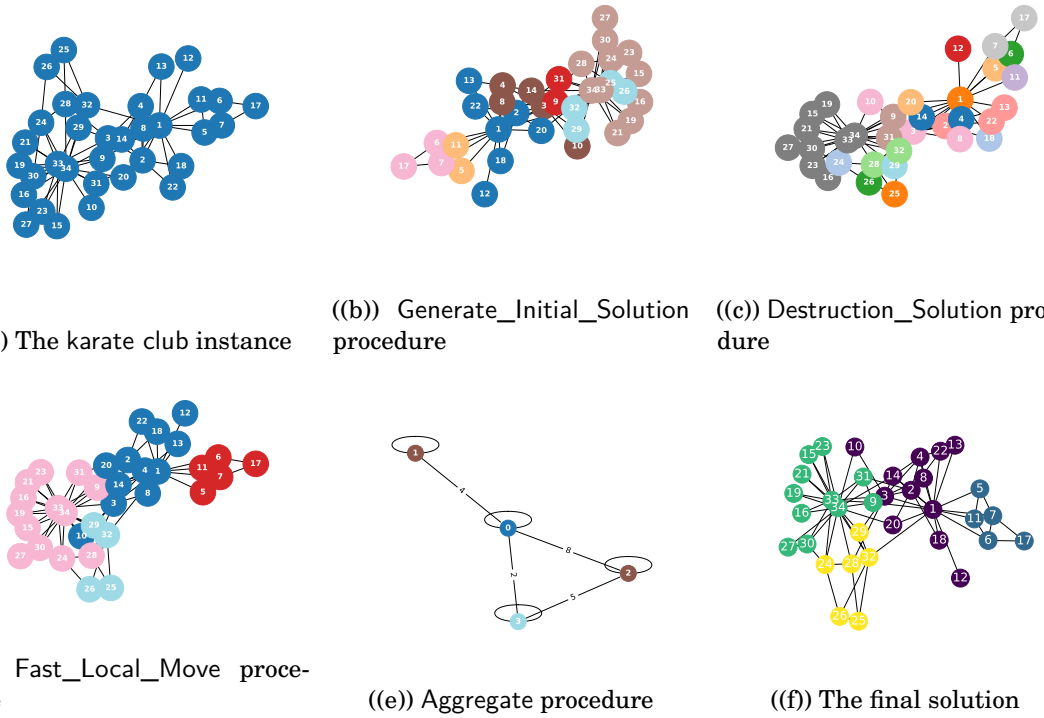
Algorithm 9 Generate_Initial_Solution procedure

Input: Graph $G(V,E)$
Output: $S = \{C_1, C_2, C_3, \ldots, C_k\}$

1: $S_0 \leftarrow \phi$
2: **for** each node $v \in V$ **do**
3: Create a new community C_v containing a single node v.
4: $S_0 \leftarrow S_0 \cup \{C_v\}$
5: **end for**
6: $S \leftarrow$Fast_Local_Move(S_0, V) ▷ See algorithm 4
7: **Return** S

5.3.2 Fast_Local_Move procedure

The Fast_Local_Move procedure (Algorithm 4.3.4) is integrated into the greedy heuristic to construct solutions, both from scratch and during the reconstruction of partial so-



((a)) The karate club instance

((b)) Generate_Initial_Solution procedure

((c)) Destruction_Solution procedure

((d)) Fast_Local_Move procedure

((e)) Aggregate procedure

((f)) The final solution

Figure 5.1: A visual representation of the individual elements of the FLMIG algorithm, demonstrated on the Karate Club network.

lutions. To promote diversity, the method incorporates randomization. As detailed in Algorithm 4, the set L is initialized with all nodes from V for building an initial solution, but is limited to the removed nodes when rebuilding a partial solution.

5.3.3 Destruction_Solution procedure

The incumbent solution is encoded using a vector representation, where communities are serialized into a single vector of length n. Each element in the vector corresponds to a node in the graph, and its value denotes the community assignment for that node. A key component of our method is the destruction phase, in which a subset of nodes is removed from the current solution to promote broader exploration of the solution space. The extent of removal is controlled by the destruction rate β, where $0 < \beta < 1$. Specifically, $\lceil \beta \cdot n \rceil$ nodes are randomly selected and extracted from the solution vector. These removed nodes are collected into a set L^{dest}, identifying them as candidates likely to change their community membership during reconstruction by merging into adjacent communities.

It is important to note that the value of β significantly influences the balance between intensification and diversification. A small β may restrict exploration, while a larger

β introduces greater randomness, potentially enhancing diversity but reducing local refinement.

Algorithm 10 Destruction_Solution procedure

Input: Graph $G(V,E), S = \{C_1, C_2, C_3, \ldots, C_k\}$, and $\beta \in [0,1]$
Output: $S^{dest} = \{C_1^{dest}, C_2^{dest}, C_3^{dest}, \ldots, C_l^{dest}\}$, $l \leq k$

$L \leftarrow V$
$L^{dest} \leftarrow \emptyset$
$n \leftarrow |V|$ ▷ Number of nodes in V
for $i = 1$ to $\lceil \beta \cdot n \rceil$ **do**
    Let v and C_v be a randomly chosen node from set L and its current community, respectively.
    $L \leftarrow L \setminus \{v\}$
    $L^{dest} \leftarrow L^{dest} \cup \{v\}$
    $C_v \leftarrow C_v \setminus \{v\}$
    if (C_v is empty) **then**
        $S \leftarrow S \setminus \{C_v\}$
    end if
end for
$S^{dest} \leftarrow S$
Return S^{dest}, L^{dest}

5.3.4 Reconstruction_Heuristic procedure

As outlined in Algorithm 11, the reconstruction phase is executed in two main steps. First, the Fast_Local_Move procedure is applied to the partially destroyed solution S^{dest}, using the set L^{dest}which contains nodes removed during the destruction phaseto guide the reconstruction. While this step efficiently restores solution quality, it may produce locally disconnected communities [163], wherein not all nodes within a community are mutually reachable. Detecting and resolving such disconnections is critical for ensuring the accuracy of the community structure.

The second step involves an enhanced version of the Louvain Prune algorithm [122], which includes a refinement procedure aimed at repairing poorly connected communities. A community is deemed well-connected if every pair of nodes within it is linked by a path. The refinement procedure assesses the connectivity of each community and, if disconnections are found, splits it into well-connected subcommunities. These subcommunities are then considered for re-merging via a local move heuristic, with both modularity and connectivity reevaluated.

Following refinement, communities are aggregated into supernodes to form a compressed representation of the graph. This aggregation step merges all nodes belonging to the same community into a single supernode, with inter-community edges updated to reflect original connections. This hierarchical compression not only accelerates subsequent

processing but also ensures consistent community quality across scales.

The enhanced Louvain Prune algorithm repeats these stepsrefinement, aggregation, and modularity optimizationacross hierarchical levels until no further modularity gains are achievable.

Algorithm 11 Reconstruction_Heuristic procedure

Input: Graph $G(V,E)$, S^{dest} and L^{dest}
Output: S

$S \leftarrow$Fast_Local_Move(S^{dest},L^{dest}) ▷ See algorithm 4
repeat
  $q \leftarrow Q(G,S)$
  $S^* \leftarrow$Refinement_procedure(S) ▷ Ensures that every community is internally connected
  $G \leftarrow$Aggregate(G,S^*) ▷ Aggregate nodes into communities
  $S \leftarrow$ put each node of G in its own community
  $S \leftarrow$Fast_Local_Move(S,V) ▷ See algorithm 4
until ($q >= Q(S,G)$) ▷ No further modularity improvement
Return flat solution S ▷ Return a non-hierarchical solution, removing the aggregation

Empirical evaluations on both real and synthetic networks indicate that the refinement procedure is primarily necessary for large instances. In smaller networks, no locally disconnected communities were observed after applying Fast_Local_Move. A summary of instances where refinement was invoked is provided in Table 5.1.

Table 5.1: Networks where the refinement_procedure was invoked.

Networks	Was the Refinement_procedure invoked?
Karate	×
Dolphins	×
Polbooks	×
Football	×
lesmis	×
Adjnoun	×
Jazz	×
Metabolic	×
NetScience	×
PGP	✓
As-22july06	✓
com-DBLP	✓
com-Amazon	✓

5.3.5 Select_Next_Solution procedure

Following reconstruction, the algorithm determines whether to adopt the newly generated solution or retain the current one for the next iteration. This acceptance decision follows a criterion inspired by Simulated Annealing [158, 164]. Even when the new solution exhibits inferior quality, it may still be accepted with a specified probabilityparticularly during earlier stages of the searchto facilitate escape from local optima. This mechanism mirrors the core principles of simulated annealing, incorporating probabilistic acceptance and a gradually decreasing temperature parameter to balance exploration and exploitation throughout the optimization process.

A new solution S^* is accepted if it has higher modularity than the current solution S, i.e., $Q(S^*) > Q(S)$. If S' is not better, it can still be accepted with a probability given by:

$$p(T,S,S^*) = \exp\left(\frac{Q(S^*) - Q(S)}{T}\right) \tag{5.1}$$

The temperature parameter T, which decreases over time according to a predefined cooling schedule, plays a central role in this simulated annealing-inspired acceptance mechanism. Following the methodology established by Stützle [158], the initial temperature is set to $T = 0.025 \cdot Q(S^{best})$, where $Q(S^{best})$ denotes the modularity of the best-known solution. A geometric cooling schedule is then applied, multiplying T by a factor of 0.9 after each iteration. As T decreases, the probability of accepting inferior solutions diminishes, shifting the search emphasis from exploration (accepting worse solutions to escape local optima) to intensification (refining high-quality solutions). This balance enhances both the diversity and final performance of the algorithm. Pseudocode for the Select_Next_Solution procedure is provided in Algorithm 12; note that temperature initialization is omitted in Algorithm 8 for readability.

Algorithm 12 Select_Next_Solution procedure

Input: The current solution S and the newly generated solution S^*.
Output: S

if $(Q(S^*) \geq Q(S))$ **then**
  $S \leftarrow S^*$
else
  $p_0 \leftarrow$ random number uniformly distributed over $[0,1]$.
  if $(p(T,S,S^*) \geq p_0)$ **then** ▷ See equation 5.1 for the definition of $p(T,S,S^*)$.
    $S \leftarrow S^*$
  end if
end if
$T \leftarrow T \times 0.9$ ▷ Reduce temperature
Return S

5.3.6 Time Complexity Analysis

This section details the time complexity analysis of the proposed FLMIG algorithm for a network with nn nodes and mm edges. The analysis covers its three core components:

- The $Fast\_Local\_Move$ procedure has a complexity of $O(n \cdot k)$, which simplifies to $O(m)$, where k is the average number of neighboring communities per node.
- The $Destruction\_Solution$ procedure operates in linear time, $O(n)$.
- The $Reconstruction\_Heuristic$, based on the Louvain Prune algorithm, also has a complexity of $O(m)$[122].

The overall time complexity of the FLMIG algorithm is determined by the sum of its components multiplied by the number of iterations (IT) in the main loop, resulting in:

$$O(m) + IT \cdot (O(n) + O(m)) = O(IT \cdot (m + n))$$

This indicates that the algorithm's runtime scales linearly with the number of iterations and the size of the network (nodes and edges).

5.4 Experimental Evaluation

5.4.1 Preview of implementation and tuning of algorithms

All experiments were executed on an Intel Core i5-13600KF CPU clocked at 5.1 GHz with 16 GB of RAM, running Ubuntu 22.04 LTS. The algorithms were implemented using the following Python packages:

- **Fast Local Move Iterated Greedy (FLMIG) and Iterated Carousel Greedy (ICG):** [1]. **Leiden (LDN):** [2]. **Louvain Prune (LVNP):** [3].

5.4.2 Problem instances

A comparative analysis was conducted to assess the efficiency of the Fast Local Move Iterated Greedy (FLMIG) algorithm relative to the Leiden (LDN) [163], Louvain Prune (LVNP) [122], and Iterated Carousel Greedy (ICG) [84] algorithms. The evaluation was performed on a suite of real-world and synthetic benchmark problems.

5.4.2.1 Real-world networks problems

First, we performed our experiments on widely recognized real-world datasets, with their characteristics outlined in Table 5.2.

[1] https://github.com/salahinfo/Iterated-greedy-algorithm-for-community-detction.git
[2] https://github.com/vtraag/leidenalg
[3] https://github.com/salahinfo/Prune_louvain-algorithm.git

Table 5.2: Overview information of real-world networks

Class	Dataset	$\|V\|$	$\|E\|$	Avg degree	Avg CC
BSRP	Karate [178]	34	78	4.588	0.588
	Dolphins [103]	62	160	5.129	0.2859
	Football [59]	115	613	10.661	0.4033
	Jazz [60]	198	2742	27.70	0.633
BMRP	Polbooks [115],	105	441	8.4	0.4875
	lesmis [82]	77	254	6.597	0.736
	Adjnoun [115]	112	425	7.589	0.190
	Metabolic [47]	453	2040	8.940	0.655
BLRP	NetScience [115]	1589	2742	3.451	0.6377
	PGP [20]	10680	24316	4.5	0.2659
	As-22july06 [117]	22963	48436	4.21	0.2304
	DBLP [92]	317080	1049866	5.530	0.732
	Amazon [92]	334863	925872	6.622	0.430

5.4.2.2 Synthetic networks problems

Synthetic experiments were conducted using Lancichinetti-Fortunato-Radicchi (LFR) benchmark networks [90]1. We generated four sets of LFR benchmarks, each containing seven distinct networks. The parameters for these benchmarks are detailed in Table 5.3. These benchmarks are organized into three main classes:

- Benchmark sets of small-scale synthetic problems (BSSP)
- Benchmark sets of moderate-scale synthetic problems (BMSP)
- Benchmark sets of large-scale synthetic problems (BLSP)

Table 5.3: The classes of synthetic problem set size

Networks	BSSP-a	BSSP-b	BMSP	BLSP
N	10^3	10^3	10^4	10^5
K_{avg}	15	15	15	15
K_{max}	100	50	50	50
$MinC$	20	10	10	10
$MaxC$	100	50	50	50

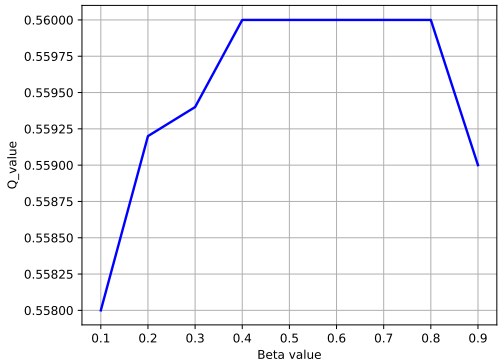
5.4.3 Algorithm tuning

The optimal performance of the FLMIG algorithm relies heavily on the effective tuning of parameters β and nbr,the source code of the FLMIG algorithm is available at https://github.com/salahinfo/FLMIG_algorithm.

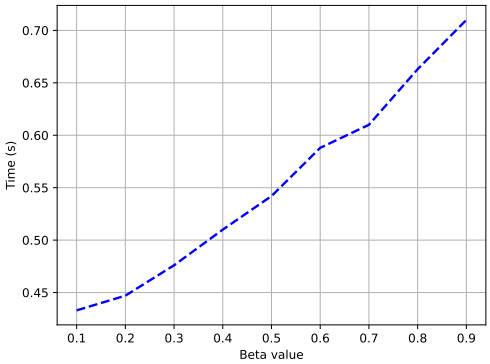
To determine optimal parameter values, we performed extensive testing on multiple real-world networks. This process was instrumental in identifying the most effective settings for the destruction rate β, with results showing consistent performance across diverse networks. For instance, we evaluated the Lesmis network under β values ranging from 0.1 to 0.9, analyzing their impact on both modularity and computational time. For each β configuration, we recorded the average modularity to quantify its effect.

A termination criterion r was established to limit the maximum number of iterations nbr without modularity improvement. The value of r was set according to network size: 100 for small networks, 50 for medium-sized networks (e.g., Netscience, PGP, As-22july06), and 10 for larger networks (e.g., DBLP, Amazon).

The influence of different β values is illustrated in Figures 5.2(a) and 5.2(b). Detailed parameter configurations for each network are provided in Table 5.4, while parameters for comparative methods are listed in Table 5.5. This systematic parameter analysis significantly enhanced the performance of the FLMIG algorithm.



((a)) The impact of β values on the modularity value.



((b)) The impact of β values on the computational time

5.5 Results and Discussion

5.5.1 Performance on real-world networks

A comparative analysis was performed to assess the efficiency of the FLMIG algorithm relative to the LDN, LVNP, and ICG algorithms in solving the BSP, BMP, and BLP

Table 5.4: Calibration of the FLMIG algorithm's parameters.

Networks	Parameters and Values
Karate	$\beta = 0.5$, $nbr = 100$
Dolphins	$\beta = 0.7$, $nbr = 100$
Football	$\beta = 0.5$, $nbr = 200$
Polbooks	$\beta = 0.4$, $nbr = 200$
Adjnoun	$\beta = 0.5$, $nbr = 300$
Lesmis	$\beta = 0.4$, $nbr = 200$
Metabolic	$\beta = 0.5$, $nbr = 300$
Netscience	$\beta = 0.5$, $nbr = 100$
PGP	$\beta = 0.5$, $nbr = 100$
As-06jully22	$\beta = 0.5$, $nbr = 100$
com-Amazon	$\beta = 0.4$, $nbr = 20$
com-dblp	$\beta = 0.4$, $nbr = 20$

Table 5.5: The established parameter sets for the comparative algorithms.

Algorithm	Parameters	Value	Description
ICG	β	0.5	Destruction rate
	nbr	300	Number of iteration
	P_c	200	Initial population

problems. The evaluation metrics included the maximum, mean, and standard deviation of the modularity value (Q) obtained from 10 independent trials.

The findings for the BSRP, BMRP, and BLRP problem sets are summarized in Tables 5.6, 5.7, and 5.8. Each table presents the aforementioned Q-value statistics for every dataset and algorithm, concluding with the computational time. The statistical comparison involved a two-step procedure: the Kruskal-Wallis test initially determined if significant differences existed among the algorithms across datasets. Following a significant result, Dunns test with Bonferroni adjustment was conducted to identify the specific pairwise differences, thereby offering a detailed performance comparison.

5.5.1.1 Performance on small-scale real-world problems (BSRP).

As shown in Table 5.6, FLMIG consistently outperformed the other algorithms on the smaller problem sets. It recorded the best modularity values for the Karate, Dolphins, Football, and Jazz datasets, with minimal variability between runs, confirming its stable and effective performance. LDN closely matched FLMIG's top scores with high

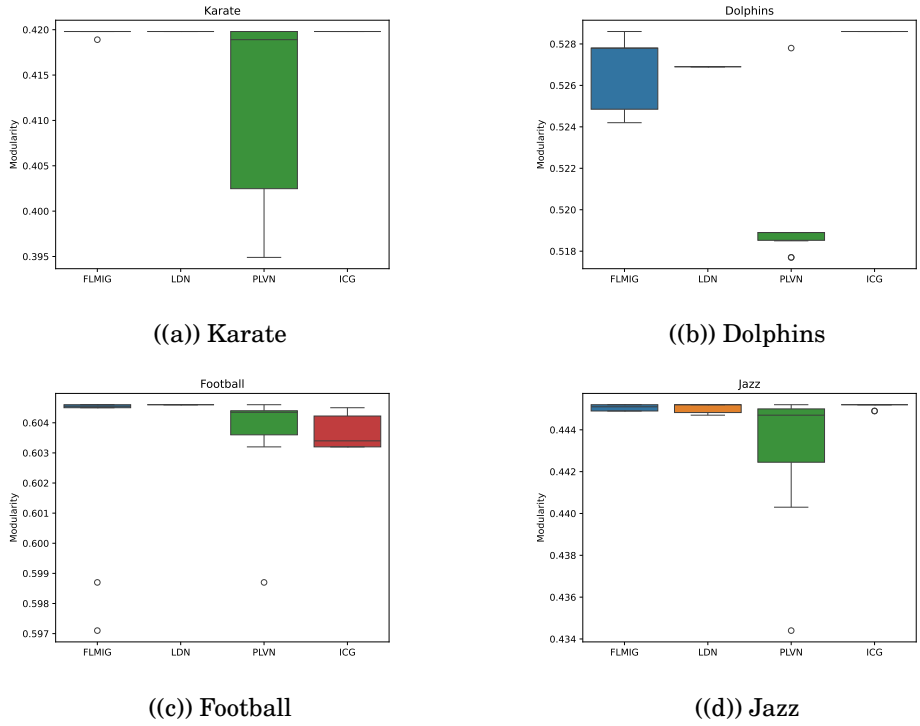
Table 5.6: Algorithm performance on small real-world problems (BSRP). Results show the Best, Avg, and Std for FLMIG, LDN, LVNP, and ICG. The best values for Best and Avg are highlighted per case.

Algorithm		Karate	Dolphins	Football	Jazz
FLMIG	BEST	**0.419800**	**0.528600**	**0.604600**	**0.445200**
	AVG	**0.419718**	0.526873	0.603355	0.445073
	STD	0.000271	0.001781	0.002721	0.000142
	Time	0.0035	0.0081	0.0171	0.0864
LDN	BEST	**0.419800**	**0.526900**	**0.604600**	**0.445200**
	AVG	**0.419800**	**0.526900**	**0.604600**	0.445064
	STD	0	0	0	0.000234
	Time	0.0033	0.0061	0.0174	0.01130
LVNP	BEST	**0.419800**	**0.527800**	**0.604600**	**0.445200**
	AVG	0.412791	0.520236	0.603709	0.443264
	STD	0.010817	0.003767	0.001728	0.003357
	Time	0.0013	0.0024	0.0074	0.0687
ICG	BEST	**0.419800**	**0.528600**	0.604500	**0.445200**
	AVG	**0.419800**	**0.528600**	0.603727	**0.445145**
	STD	0	0	0.000618	0.000121
	Time	3.4006	6.2010	11.5020	19.8067
K-W Test	H-statistics	17.2094	30.0237	18.8160	11.4031
	P-value	0.000600	0.000001	0.000200	0.009700
DcB Test	FLMIG-LDN	1	1	0.346400	1
	FLMIG-LVNP	0.015900	0.030000	1	0.249700
	FLMIG-ICG	1	0.040000	0.185800	1
	LDN-LVNP	0.002500	0.159400	0.007200	0.080200
	LDN-ICG	1	0.007500	0.000300	1
	LVNP-ICG	0.002500	0	1	0.007600

consistency, though its average results were slightly lower and it was computationally slower. Conversely, LVNP delivered weaker results than both, particularly on the Karate and Dolphins datasets. However, LVNP's results showed higher variability. Its principal advantage was speed, as it achieved the lowest execution times. ICG produced mixed outcomes; while it matched FLMIG's accuracy on Karate and Dolphins, its performance dropped on Football. Furthermore, despite its competitive accuracy, ICG was the slowest algorithm by a significant margin, reducing its overall efficiency.

Statistical Significance: The Kruskal-Wallis test validated significant differences in algorithm performance. Subsequent analysis (Dunns test with Bonferroni correction) showed that FLMIG and LDN are statistically superior to LVNP and ICG.

Conclusion: For smaller datasets, FLMIG is the recommended algorithm due to



((a)) Karate

((b)) Dolphins

((c)) Football

((d)) Jazz

Figure 5.3: Performance on small-scale real-world problems (BSRP).

its optimal balance of high accuracy and efficiency. LDN is a strong, slightly slower alternative. LVNP prioritizes speed, and ICG is ineffective due to its high computational cost. See Figure 5.3 for a visual summary. The execution time of FLMIG is quite low compared to the other algorithms, making it an efficient choice.

5.5.1.2 Performance on moderate-scale real-world problems (BMRP).

The evaluation on moderate-sized problem sets, as shown in Table 5.7, establishes FLMIG's strong performance. The algorithm secured the optimal result on three out of four datasets with minimal deviation, demonstrating robust performance. A slight increase in variability was observed solely on the Adjnoun dataset.

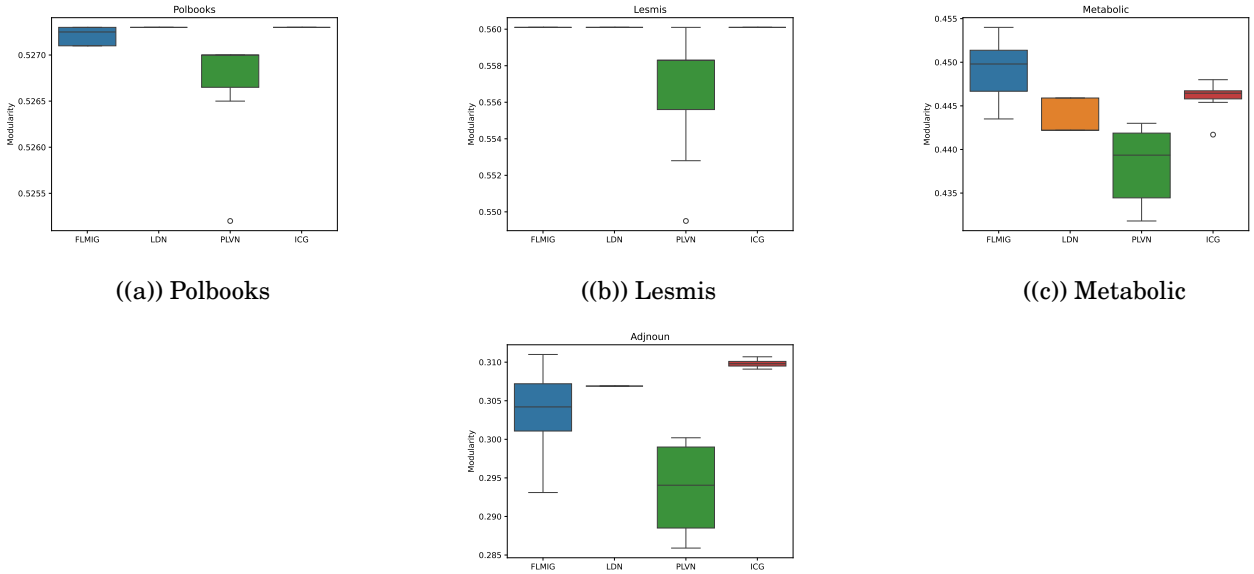
LDN closely matched FLMIG's performance on the Polbooks and Lesmis datasets but showed a slight decline on the more complex Metabolic and Adjnoun networks. Despite this, it maintained stable results and fast computation. In contrast, LVNP consistently yielded the weakest accuracy and highest result variability, though it was the fastest algorithm. While ICG produced solid resultseven rivaling FLMIG's performance on the Polbooks and Lesmis datasetsit was consistently slower than the other algorithms. Its slightly lower accuracy on the Metabolic and Adjnoun problems, combined with this high computational cost, diminished its overall competitiveness.

Table 5.7: Algorithm performance on moderate-scale real-world problems (BMRP). Results (Best, Avg, Std) for FLMIG, LDN, LVNP, and ICG. The top-performing values for Best and Avg in each case are highlighted.

Algorithm		Polbooks	Lesmis	Metabolic	Adjnoun
FLMIG	BEST	**0.527300**	**0.560100**	**0.454000**	**0.311000**
	AVG	0.527218	**0.560100**	**0.449573**	0.304227
	STD	0.000098	0	0.003775	0.005929
	Time	0.0170	0.0074	0.1718	0.0175
LDN	BEST	**0.527300**	**0.560100**	0.445900	0.306900
	AVG	**0.527300**	**0.560100**	0.443882	0.306900
	STD	0	0	0.001932	0
	Time	0.0141	0.0101	0.0182	0.0146
LVNP	BEST	0.527000	**0.560100**	0.443000	0.300200
	AVG	0.526736	0.556836	0.438836	0.294209
	STD	0.000541	0.003243	0.004311	0.005782
	Time	0.0060	0.0033	0.0844	0.0094
ICG	BEST	**0.527300**	**0.560100**	0.448000	0.310700
	AVG	**0.527300**	**0.560100**	0.446200	**0.309918**
	STD	0	0	0.001704	0.000578
	Time	10.5003	7.7002	4.5321	11.2001
K-W Test	H-statistics	32.8446	33.3755	26.7468	28.5156
	P-value	0	0	0.000007	0.000003
DcB Test	FLMIG-LDN	0.588200	1	0.017700	1
	FLMIG-LVNP	0.005600	0.000014	0.000007	0.109000
	FLMIG-ICG	0.588200	1	0.000014	0.020200
	LDN-LVNP	0.000004	0.000014	0.346200	0.013000
	LDN-ICG	1	1	0.549500	0.155700
	LVNP-ICG	0.000004	0.000014	0.002000	0.000001

Statistical Analysis: The Kruskal-Wallis test confirmed statistically significant differences in algorithm performance across all datasets ($p \simeq 0$). Post-hoc analysis (Dunn's test with Bonferroni correction) revealed that FLMIG and LDN form a statistically superior tier, consistently outperforming LVNP and ICG.

Conclusion: For moderate-sized problems, FLMIG is the optimal algorithm, leading in both accuracy and efficiency. LDN serves as a strong, high-performance alternative. The remaining algorithms represent a clear trade-off: LVNP is the fastest but least accurate, while ICG provides competitive accuracy at a prohibitively high computational cost. These results are summarized visually in Figure 5.4.



((a)) Polbooks

((b)) Lesmis

((c)) Metabolic

((d)) Adjnoun

Figure 5.4: Performance on moderate-scale real-world problems (BMRP).

5.5.1.3 Performance on large-scale real-world problems (BLRP).

As shown in the table 5.8, LDN demonstrated superior performance for large-scale problems, delivering the highest accuracy on most datasets (4/5) with scalable execution times. FLMIG was a strong competitor, providing top-tier, stable results on two datasets(Netscience and Dblp), but was hampered by a substantial increase in computation time for the largest instances. LDN also showed high consistency with low result variability. In contrast, LVNP traded accuracy for speed, delivering the fastest execution times but with significantly weaker results. ICG was the least effective overall, exhibiting the poorest accuracy and prohibitively long runtimes that render it unsuitable for large-scale use.

Statistical Analysis: The Kruskal-Wallis test confirms statistically significant performance differences among the algorithms ($p \simeq 0$). Post-hoc analysis (Dunn's test with Bonferroni correction) reveals that FLMIG and LDN form a statistically superior group, consistently outperforming LVNP and ICG on larger datasets. Furthermore, LDN demonstrates a significant performance advantage over LVNP, particularly on the as-22july06, Amazon, and Dblp datasets.

Conclusion: For large-scale problems, LDN emerges as the optimal algorithm, successfully combining high accuracy with computational efficiency. FLMIG remains a strong alternative in terms of accuracy but is less efficient due to its higher computational cost. The results present a clear trade-off: LVNP offers speed at the expense of accuracy,

Table 5.8: Algorithm performance on large-scale real-world problems (BLRP). Results (Best, Avg, Std) for FLMIG, LDN, LVNP, and ICG. The best values for Best and Avg are highlighted per case.

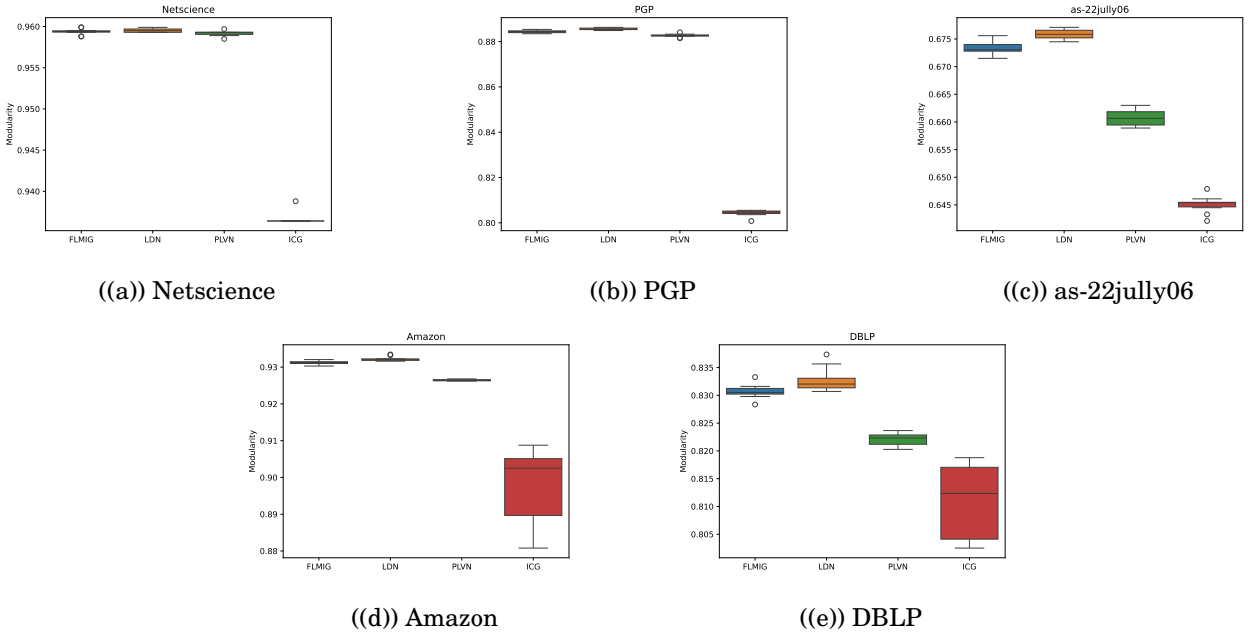
Algorithm		Netscience	PGP	as-22jully06	Amazon	Dblp
FLMIG	BEST	**0.959900**	0.885400	0.675600	0.932050	**0.833280**
	AVG	0.959427	0.884509	0.673536	0.931228	0.830691
	STD	0.000385	0.000632	0.001296	0.000471	0.001280
	Time	0.1641	1.3026	11.2710	284.6380	723.3649
LDN	BEST	**0.959900**	**0.886300**	**0.677100**	**0.933500**	0.833200
	AVG	**0.959555**	**0.885655**	0.676000	**0.932382**	**0.832145**
	STD	0.000254	0.000520	0.000914	0.000695	0.000832
	Time	0.1280	0.8688	1.7882	15.0941	36.6919
LVNP	BEST	0.959700	0.884100	0.663000	0.926800	0.823700
	AVG	0.959227	0.882827	0.660900	0.926536	0.822318
	STD	0.000341	0.000822	0.001570	0.000191	0.001186
	Time	0.0418	0.5455	5.9267	30.7425	70.0546
ICG	BEST	0.938800	0.805500	0.647900	0.908800	0.818200
	AVG	0.936836	0.804455	0.645327	0.898273	0.810891
	STD	0.000971	0.001359	0.001707	0.011002	0.006685
	Time	158.9021	1068.1591	2296.5426	3440.7619	3140.1616
K-W Test	H-statistics	26.1430	35.3509	35.8583	35.7630	36.7109
	P-value	0.000009	0	0	0	0
DcB Test	FLMIG-LDN	1	1	0.531200	0.701948	1
	FLMIG-LVNP	0.109000	0.897500	0.261000	0.306845	0.216433
	FLMIG-ICG	0.020200	0.000100	0.000500	0.000217	0.000110
	LDN-LVNP	0.013000	0.519500	0.001186	0.002611	0.004477
	LDN-ICG	0.155700	0.000031	0	0	0
	LVNP-ICG	0.000001	0.026800	0.334000	0.217490	0.217490

while ICG is non-competitive due to its poor performance and prohibitive execution times. A visual summary is provided in Figure 5.5.

5.5.2 Performance on synthetic networks problems

To further evaluate the algorithms' effectiveness, we conducted scalability tests on the BSSP-a, BSSP-b, BMSP, and BLSP problem sets. Similar to the previous experiments, performance was measured using modularity, reporting the best, average, and standard deviation values over 10 independent runs to assess effectiveness and stability.

The primary scaling parameter was the mixing parameter (μ), which was increased from 0.1 to 0.8 in increments of 0.1, creating eight distinct test cases. All other para-



((a)) Netscience ((b)) PGP ((c)) as-22jully06

((d)) Amazon ((e)) DBLP

Figure 5.5: Performance on large-scale real-world problems (BLRP).

meters remained consistent with the initial experimental setup (see Section 4.3). The results of these scalability experiments are detailed in Tables 5.9, 5.10, 5.11, and 5.12. In these tables, the additional row labeled 'ţ' indicates the value of the mixing parameter for each test case.

5.5.2.1 Performance on small-scale synthetic problems (BSSP-a).

The performance evaluation of the four algorithms (FLMIG, LDN, LVNP, ICG) on the smaller synthetic problem set (BSSP-a) is presented in Table 5.9. The table details the best and average modularity, standard deviation, and execution time for each algorithm across eight values of the mixing parameter $\mu(0.1to0.8)$.

FLMIG delivered the best overall accuracy, achieving top scores with perfect consistency (zero standard deviation) until $\mu = 0.5$. However, its computational cost grew substantially with problem size, peaking at 0.9277 seconds for the largest instance.

LDN matched FLMIG's accuracy but exhibited rising result instability from ţ=0.6, offset by significantly faster execution. LVNP showed a similar but less accurate trend than LDN, with stability declining from $\mu = 0.5$. ICG was the least effective, with the lowest scores, high variability, and prohibitively long runtimes.

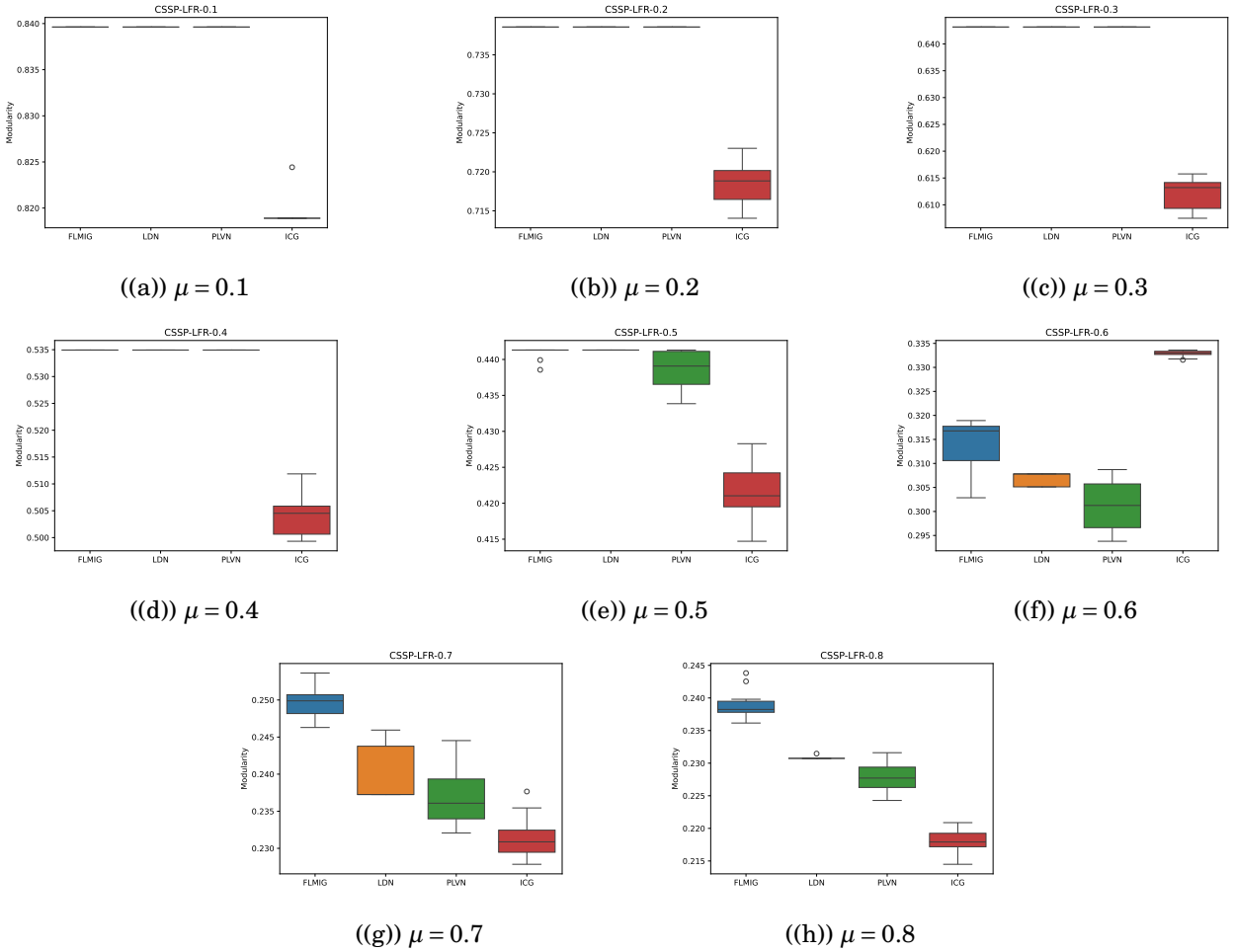
Statistical Analysis: The Kruskal-Wallis test confirms statistically significant performance differences among the algorithms ($P - value = 0$). Post-hoc analysis (Dunn's test with Bonferroni correction) validates these findings, revealing that FLMIG signifi-

cantly outperforms ICG ($P - value \simeq 0$). While FLMIG and LDN show more comparable performance, statistically significant differences exist between all algorithm pairs, a distinction that becomes more pronounced with larger problem sizes.

Conclusion: In summary, FLMIG achieves the highest accuracy but requires the greatest computational time. LDN presents a favorable balance of competitive accuracy and superior speed. LVNP offers faster execution but with reduced stability, while ICG is the least effective algorithm overall. These results are visualized in Figure 5.6.

Table 5.9: Algorithm performance on small synthetic problems (BSSP-a). Results show the Best, Avg, and Std for FLMIG, LDN, LVNP, and ICG. The best values for Best and Avg are highlighted per case.

Algorithm		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
FLMIG	Best	**0.839626**	**0.738568**	**0.643158**	**0.534936**	**0.441307**	**0.318925**	**0.253626**	**0.243817**
	Avg	**0.839626**	**0.738568**	**0.643158**	**0.534936**	0.440893	**0.313826**	**0.249834**	**0.238976**
	Std	0	0	0	0	0.000929	0.005563	0.002396	0.002447
	Time	0.1961	0.2113	0.2412	0.2783	0.3247	0.5701	0.8736	0.9277
LDN	Best	**0.839626**	**0.738568**	**0.643158**	**0.534936**	**0.441307**	0.307815	0.245941	0.231467
	Avg	**0.839626**	**0.738568**	**0.643158**	**0.534936**	**0.441307**	0.306736	0.239857	0.230813
	Std	0	0	0	0	0	0.001393	0.004198	0.000230
	Time	0.0577	0.0576	0.0600	0.0582	0.0644	0.0709	0.0769	0.0810
LVNP	Best	**0.839626**	**0.738568**	**0.643158**	**0.534936**	**0.441307**	0.308713	0.244525	0.231606
	Avg	**0.839626**	**0.738568**	**0.643158**	**0.534936**	0.438452	0.301291	0.236869	0.227867
	Std	0	0	0	0	0.002809	0.005486	0.003760	0.002262
	Time	0.1222	0.1300	0.1548	0.1779	0.2093	0.2431	0.2896	0.3339
ICG	Best	0.824425	0.723015	0.615761	0.511867	0.428280	0.333609	0.237661	0.220880
	Avg	0.819451	0.718738	0.612248	0.504382	0.421417	0.332860	0.231470	0.218145
	Std	0.001748	0.002922	0.003079	0.004203	0.004609	0.000701	0.003160	0.001875
	Time	100.0765	101.0765	102.0765	103.0765	104.0765	105.0765	106.0765	105.0870
K-W Test	H-statistics	38.7097	37.9624	37.9624	37.9562	31.1414	30.3814	30.2007	35.6698
	P-value	0	0	0	0	0.000001	0.000001	0.000001	0
DcB Test	FLMIG-LDN	1	1	1	1	1	0.572372	0.057672	0.206096
	FLMIG-LVNP	1	1	1	1	0.539340	0.024229	0.002499	0.001544
	FLMIG-ICG	0.000002	0.000003	0.000003	0.000003	0.000037	0.122271	0	0
	LDN-LVNP	1	1	1	1	0.148968	1	1	0.743016
	LDN-ICG	0.000002	0.000003	0.000003	0.000003	0.000002	0.000402	0.030616	0.001544
	LVNP-ICG	0.000002	0.000003	0.000003	0.000003	0.028530	0.000001	0.376811	0.206096



((a)) $\mu = 0.1$ ((b)) $\mu = 0.2$ ((c)) $\mu = 0.3$

((d)) $\mu = 0.4$ ((e)) $\mu = 0.5$ ((f)) $\mu = 0.6$

((g)) $\mu = 0.7$ ((h)) $\mu = 0.8$

Figure 5.6: Performance on small-scale synthetic problems (BSSP-a).

5.5.2.2 Performance on small-scale synthetic problems (BSSP-b).

The performance of the four algorithms (FLMIG, LDN, LVNP, ICG) on the smaller synthetic problem set BSSP-b is summarized in Table 5.10. The comparison is conducted across a range of mixing parameters ($\mu = 0.1 to 0.8$) and evaluates the best and average modularity, standard deviation, and execution time.

FLMIG is the top-performing algorithm in terms of accuracy, delivering high best and average scores with exceptional consistency. A key trade-off is its execution time, which grows with problem complexity. Despite being slower than LDN and LVNP, it offers a robust balance between performance and efficiency. LDN is a close competitor to FLMIG in accuracy but is consistently faster, though its performance gap widens slightly on larger problems. LVNP offers a middle ground: it is faster than FLMIG but slower than LDN, and delivers lower accuracy with decreasing stability as problem size increases. ICG is the least accurate algorithm, particularly on larger problems. Although

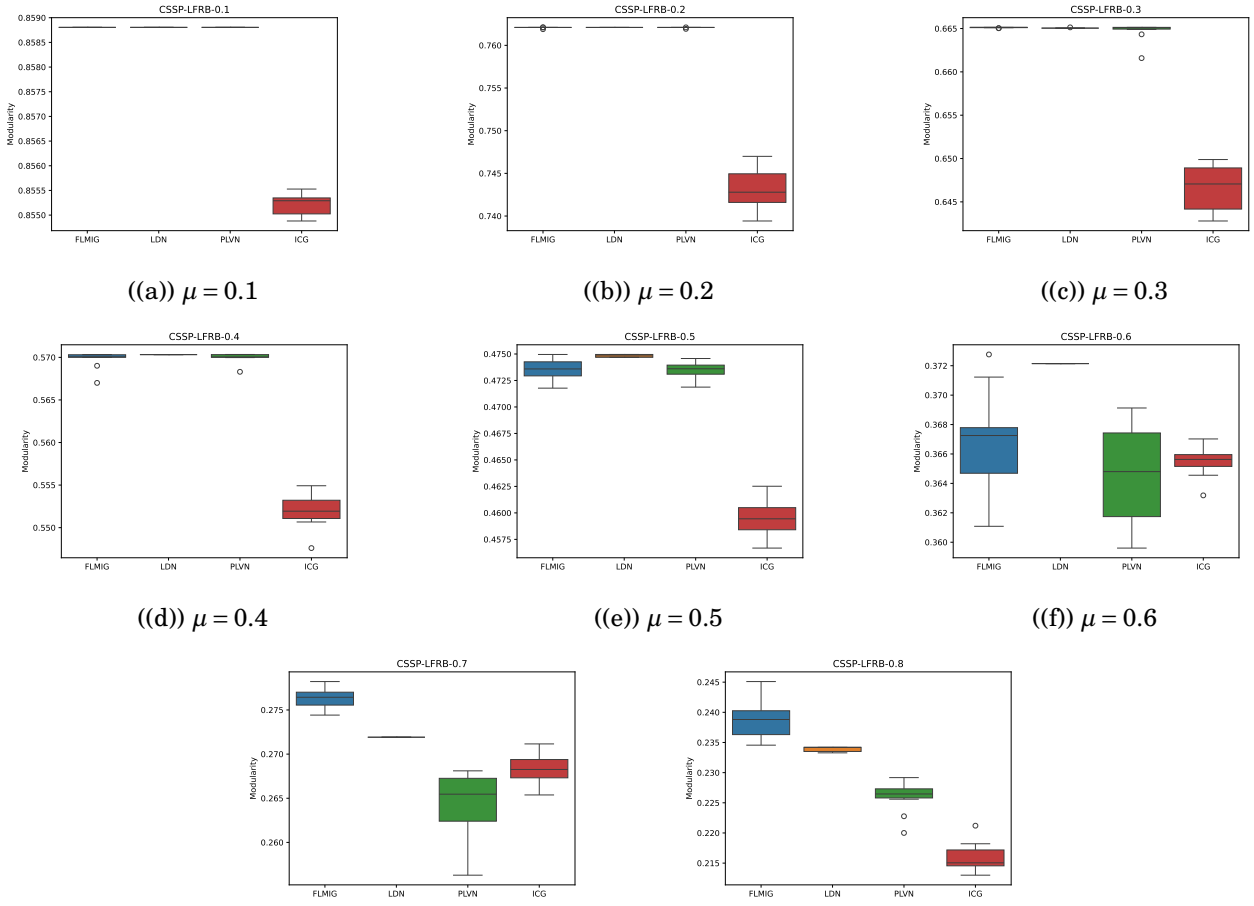
sometimes more stable than LVNP, it is computationally inefficient, with execution times exceeding 100 seconds, rendering it impractical for large-scale use. ă

Statistical Analysis: The Kruskal-Wallis test confirms statistically significant performance differences among the algorithms ($P - value \simeq 0$). Post-hoc analysis (Dunn's test with Bonferroni correction) substantiates these findings, demonstrating that FLMIG consistently outperforms LDN, LVNP, and ICG. Although the performance gap between FLMIG and LDN is narrower in some instances, the differences remain statistically meaningful.

Conclusion: Based on the results, FLMIG is identified as the most accurate algorithm. LDN is the optimal choice for speed, delivering the fastest execution times while maintaining competitive accuracy. LVNP offers a middle ground but suffers from diminishing stability at larger scales. ICG is the least viable option due to its low accuracy and high computational cost, despite showing relative stability. The results are summarized graphically in Figure 5.7.

Table 5.10: Algorithm performance on small-scale synthetic problems (BSSP-b). Results (Best, Avg, Std) for FLMIG, LDN, LVNP, and ICG. The best values for Best and Avg are highlighted per case.

Algorithm		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
FLMIG	Best	**0.858805**	**0.762107**	**0.665136**	0.570323	**0.474956**	**0.372765**	**0.278217**	**0.245098**
	Avg	**0.858805**	0.762081	**0.665113**	0.569743	0.473554	0.366934	**0.276318**	**0.239118**
	Std	0	0.000066	0.000040	0.001038	0.000955	0.003463	0.001289	0.003618
	Time	0.1971	0.2112	0.2396	0.2765	0.3313	0.4842	0.7575	0.8969
LDN	Best	**0.858805**	0.762100	**0.665136**	0.570323	**0.474956**	0.372143	0.271915	0.234195
	Avg	**0.858805**	0.762100	0.665057	**0.570323**	**0.474821**	**0.372143**	0.271915	0.233921
	Std	0	0	0.000028	0	0.000142	0	0	0.000441
	Time	0.0577	0.0576	0.0600	0.0582	0.0644	0.0709	0.0769	0.0810
LVNP	Best	**0.858805**	**0.762107**	**0.665136**	**0.570341**	0.474574	0.369127	0.268105	0.229174
	Avg	**0.858805**	**0.762084**	0.664660	0.569969	0.473473	0.364591	0.264248	0.225876
	Std	0	0.000054	0.001107	0.000606	0.000856	0.003348	0.003923	0.002631
	Time	0.1318	0.1519	0.1669	0.1987	0.2282	0.2896	0.3156	0.3337
ICG	Best	0.855528	0.746994	0.649884	0.554922	0.462516	0.367022	0.271155	0.221206
	Avg	0.855211	0.743145	0.646670	0.551989	0.459599	0.365536	0.268267	0.215962
	Std	0.000219	0.002306	0.002722	0.002146	0.001871	0.001126	0.001700	0.002415
	Time	100.0345	101.0345	102.0345	103.0345	104.0345	105.0345	106.0345	106.6540
K-W Test	H-statistics	29.1648	26.0862	26.0862	26.3369	31.5534	21.6483	35.2654	36.6468
	P-value	0.000002	0.000009	0.000009	0.000008	0.000001	0.000077	0	0
DcB Test	FLMIG-LDN	1	0.507196	0.507196	0.421602	0.064538	0.037162	0.323349	0.330563
	FLMIG-LVNP	1	1	1	1	1	1	0	0.000693
	FLMIG-ICG	0.000026	0.000007	0.000007	0.009689	0.013338	1	0.000204	0
	LDN-LVNP	1	1	1	0.966506	0.033525	0.000310	0.002173	0.316239
	LDN-ICG	0.000107	0.000107	0.010614	0.000004	0	0.000365	0.159795	0.000810
	LVNP-ICG	0.000097	0.000097	0.000929	0.002208	0.027224	1	1	0.360834



((a)) $\mu = 0.1$ ((b)) $\mu = 0.2$ ((c)) $\mu = 0.3$

((d)) $\mu = 0.4$ ((e)) $\mu = 0.5$ ((f)) $\mu = 0.6$

((g)) $\mu = 0.7$ ((h)) $\mu = 0.8$

Figure 5.7: Performance on small-scale synthetic problems (BSSP-b)

5.5.2.3 Performance on moderate-scale synthetic problems (BMSP).

Table 5.11 compares the performance of FLMIG, LDN, LVNP, and ICG on the moderate-size synthetic problem set (BMSP) across a range of mixing parameters ($\mu = 0.1 to 0.8$), reporting best and average values, standard deviation, and execution time.

FLMIG achieves the best accuracy with high stability but has a significant computational cost that grows with problem size. It is ideal for applications where accuracy is the highest priority, but less suitable for time-sensitive tasks. LDN delivers accuracy nearly identical to FLMIG but with significantly faster speed, making it a balanced and efficient choice, especially for larger datasets. In contrast, LVNP performs well on small problems but its accuracy and stability decline as size increases. It is faster than FLMIG but slower than LDN, positioning it as a mid-tier option. ICG is the least effective algorithm, demonstrating significantly lower accuracy and prohibitively long execution times that exceed 1000 seconds, rendering it impractical for larger problems. ă

Statistical Analysis Statistical analysis confirms the significance of the observed performance differences. The Kruskal-Wallis test yields low P-values, rejecting the null hypothesis of equal algorithm performance. Post-hoc testing (Dunn's test with Bonferroni correction) clarifies these differences, showing that FLMIG significantly outperforms ICG. Meanwhile, the performance between LDN and LVNP is more comparable, with statistically minimal differences in several cases.

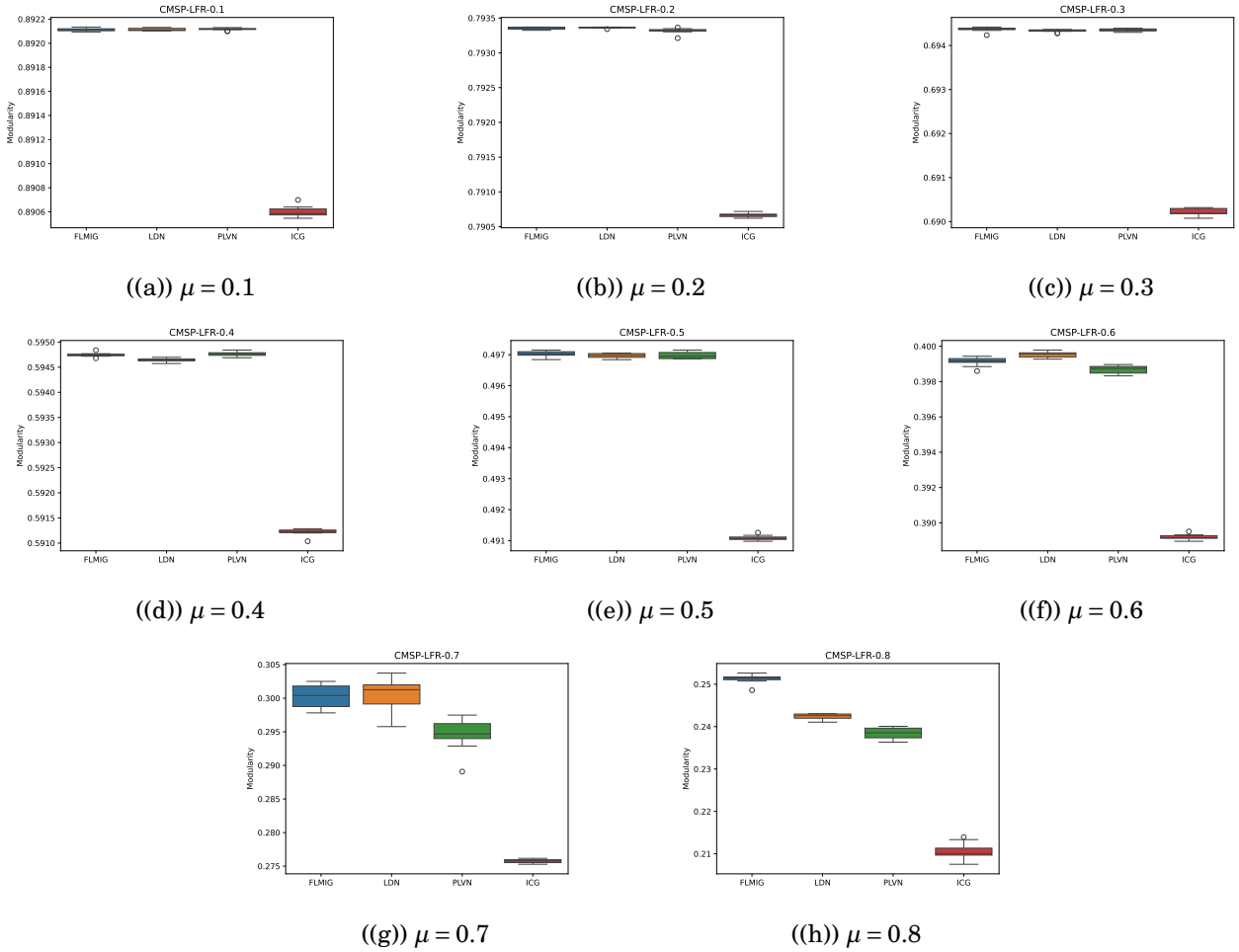
Conclusion In summary, the results indicate distinct application scenarios for each algorithm:

- FLMIG is optimal for applications demanding maximum accuracy.
- LDN provides the best balance of high performance and computational efficiency.
- LVNP is a viable option for smaller-scale problems where speed is prioritized.
- ICG is the least viable due to its poor accuracy and prohibitively long execution times.

These relationships are visualized in Figure 5.8.

Table 5.11: Algorithm performance on moderate-scale synthetic problems (BMSP). Results (Best, Avg, Std) for FLMIG, LDN, LVNP, and ICG. The best values for Best and Avg are highlighted per case.

Algorithm		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
FLMIG	Best	**0.892135**	**0.793371**	**0.694420**	0.594839	**0.497151**	0.399434	0.302507	**0.252637**
	Avg	0.892112	0.793355	**0.694368**	0.594748	**0.497028**	0.399127	0.300336	**0.251290**
	Std	0.000012	0.000018	0.000051	0.000041	0.000091	0.000248	0.001754	0.001116
	Time	2.247	2.8944	3.4605	4.0468	5.2258	7.6676	11.528	17.7839
LDN	Best	0.892131	**0.793371**	0.694366	0.594699	0.497054	**0.399784**	**0.303745**	0.243058
	Avg	**0.892113**	**0.793362**	0.694333	0.594639	0.496971	**0.399522**	**0.300638**	0.242340
	Std	0.000011	0.000008	0.000031	0.000040	0.000076	0.000155	0.002333	0.000742
	Time	0.593	0.6397	0.6915	0.7147	0.7703	0.7903	0.8478	0.8751
LVNP	Best	0.892130	0.793365	0.694393	**0.594842**	0.497150	0.398965	0.297490	0.240068
	Avg	0.892117	0.793317	0.694352	**0.594762**	0.496974	0.398679	0.294589	0.238475
	Std	0.000010	0.000041	0.000031	0.000046	0.000108	0.000240	0.002379	0.001334
	Time	1.4964	2.0128	2.4925	3.0064	3.7333	4.4217	5.252	5.1169
ICG	Best	0.890699	0.790719	0.690318	0.591281	0.491258	0.389512	0.276167	0.213935
	Avg	0.890602	0.790666	0.690217	0.591216	0.491083	0.389196	0.275750	0.210588
	Std	0.000044	0.000030	0.000080	0.000069	0.000084	0.000155	0.000306	0.001906
	Time	1000.0765	1001.0765	1002.0765	1003.0765	1004.0765	1005.0765	1006.0765	1006.0785
K-W Test	H-statistics	22.4097	28.8096	25.6872	32.6447	23.2258	34.5065	32.3430	36.5888
	P-value	0.000054	0.000002	0.000011	0	0.000036	0	0	0
DcB Test	FLMIG-LDN	1	1	0.334489	0.052652	1	0.334558	1	0.334627
	FLMIG-LVNP	1	0.143921	1	1	1	0.623732	0.047049	0.000783
	FLMIG-ICG	0.002083	0.000029	0.000007	0.000029	0.000046	0.001148	0.000022	0
	LDN-LVNP	1	0.136908	1	0.016026	1	0.002411	0.021861	0.334627
	LDN-ICG	0.001335	0.000026	0.019314	0.306223	0.002592	0.000000	0.000006	0.000783
	LVNP-ICG	0.000154	0.123767	0.001439	0.000004	0.002992	0.212166	0.292890	0.334627



((a)) $\mu = 0.1$ ((b)) $\mu = 0.2$ ((c)) $\mu = 0.3$

((d)) $\mu = 0.4$ ((e)) $\mu = 0.5$ ((f)) $\mu = 0.6$

((g)) $\mu = 0.7$ ((h)) $\mu = 0.8$

Figure 5.8: Performance on moderate-scale synthetic problems (BMSP).

5.5.2.4 Performance on large-scale synthetic problems (BLSP)

Table 5.12 compares the performance of FLMIG, LDN, LVNP, and ICG on the large-size synthetic problem set (BLSP) across mixing parameters from $\mu = 0.1 to 0.8$, reporting best and average values, standard deviation, and execution time.

FLMIG demonstrates strong and consistent accuracy, particularly on smaller problems, with low standard deviation indicating reliable results. A key observation is that its execution time grows predictably as the problem size increases, though it remains within manageable limits. LDN matches or even slightly exceeds FLMIG's accuracy with high consistency, but its execution time increases drastically on large problems, limiting its practicality. LVNP also provides competitive accuracy, though it generally lags behind both FLMIG and LDN, especially as problem size grows. LVNP is impractical for large-scale use due to its extremely high execution time, despite having reasonable accuracy. ICG is the weakest performer, with the lowest accuracy, slowest speed, and

least stable results.

Statistical Analysis: Statistical analysis confirms significant performance differences among the algorithms. The Kruskal-Wallis test yields very low P-values, rejecting the null hypothesis that the results occurred by chance. Post-hoc testing (Dunn's test with Bonferroni correction) substantiates these findings, demonstrating FLMIG's significant superiority over ICG. The performance of LDN and LVNP is more comparable, being statistically competitive in several instances.

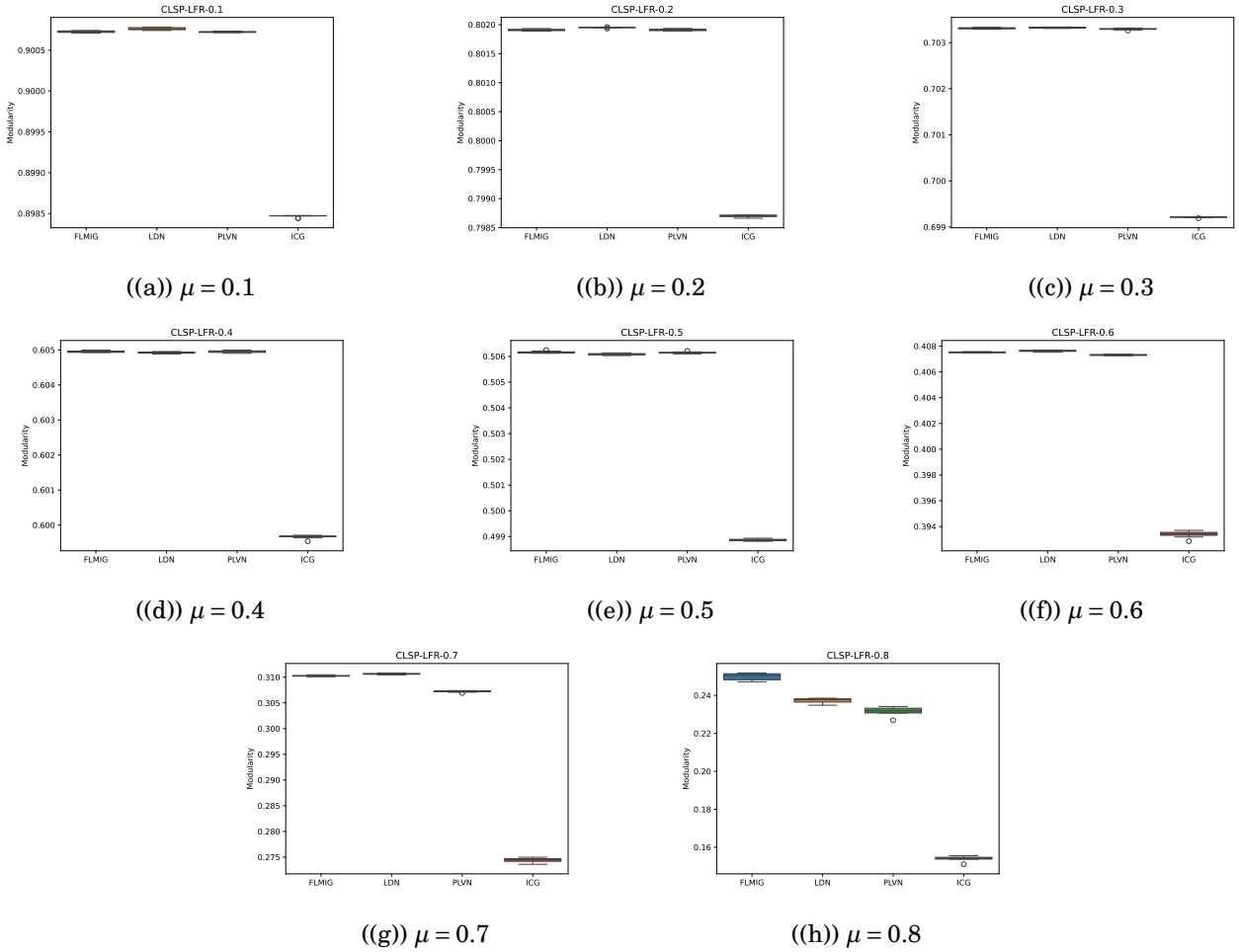
Conclusion: In summary, for large-scale synthetic problems:

- FLMIG and LDN are the top-performing algorithms in terms of accuracy, with FLMIG holding an advantage in time-efficiency.
- LVNP provides a less effective compromise, offering reduced accuracy and poor computational scalability.
- ICG is the least suitable algorithm due to its inferior accuracy and slow execution speed.

These results are visualized in Figure 5.9.

Table 5.12: Algorithm performance on large-scale synthetic problems (BLSP). Results (Best, Avg, Std) for FLMIG, LDN, LVNP, and ICG. The best values for Best and Avg are highlighted per case.

Algorithm		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
FLMIG	Best	0.900740	0.801933	0.703334	**0.604998**	**0.506247**	0.407568	0.310474	**0.251754**
	Avg	0.900726	0.801910	0.703313	**0.604952**	**0.506165**	0.407514	0.310260	**0.249895**
	Std	0.000010	0.000015	0.000014	0.000024	0.000039	0.000036	0.000127	0.001770
	Time	37.2287	61.3815	92.7853	126.394	185.8676	230.5278	315.2526	650.8977
LDN	Best	**0.900783**	**0.801960**	**0.703339**	0.604956	0.506138	**0.407691**	**0.310800**	0.238483
	Avg	**0.900762**	**0.801951**	**0.703326**	0.604922	0.506086	**0.407621**	**0.310636**	0.237278
	Std	0.000014	0.000007	0.000010	0.000021	0.000034	0.000060	0.000109	0.001227
	Time	7.7707	8.4898	8.9789	9.5936	10.2837	11.1313	12.6324	15.456
LVNP	Best	0.900733	0.801936	0.703317	0.604995	0.506213	0.407373	0.307396	0.234140
	Avg	0.900723	0.801912	0.703294	0.604952	0.506145	0.407307	0.307233	0.231671
	Std	0.000006	0.000015	0.000016	0.000031	0.000033	0.000043	0.000146	0.002141
	Time	26.0225	50.7933	80.4861	116.6926	163.9604	204.8564	270.098	277.9552
ICG	Best	0.898475	0.798715	0.699222	0.599709	0.498933	0.393703	0.274978	0.155503
	Avg	0.898466	0.798700	0.699213	0.599666	0.498865	0.393395	0.274425	0.154058
	Std	0.000013	0.000019	0.000010	0.000050	0.000039	0.000246	0.000405	0.001210
	Time	10000.0765	10001.0765	10002.0765	10003.0765	10004.0765	10005.0765	10006.0765	10005.0875
K-W Test	H-statistics	33.1070	32.7263	30.1873	25.6668	30.7068	35.6327	36.4405	36.5854
	P-value	0	0	0.000001	0.000011	0.000001	0	0	0
DcB Test	FLMIG-LDN	0.047049	0.021870	1	0.490539	0.027856	0.599883	0.365189	0.334696
	FLMIG-LVNP	1	1	0.814306	1	1	0.244172	0.320263	0.000783
	FLMIG-ICG	0.012435	0.029567	0.000670	0.000050	0.000004	0.000451	0.000725	0
	LDN-LVNP	0.012435	0.033278	0.024699	0.674307	0.183994	0.001337	0.000846	0.334696
	LDN-ICG	0	0	0.000001	0.039636	0.183994	0	0	0.000783
	LVNP-ICG	0.047049	0.019339	0.106214	0.000101	0.000092	0.334696	0.334696	0.334696



((a)) $\mu = 0.1$ ((b)) $\mu = 0.2$ ((c)) $\mu = 0.3$

((d)) $\mu = 0.4$ ((e)) $\mu = 0.5$ ((f)) $\mu = 0.6$

((g)) $\mu = 0.7$ ((h)) $\mu = 0.8$

Figure 5.9: Performance on large-scale synthetic problems (BLSP).

5.6 Conclusion

In this chapter, we presented the Fast Local Move Iterated Greedy (FLMIG) algorithm, an effective method for enhancing community detection in both synthetic and real-world networks. By integrating fast local move heuristics within an iterated greedy framework, FLMIG achieves high modularity optimization while preserving scalability and implementation simplicity. Extensive experiments confirm that FLMIG consistently identifies high-quality community structures across diverse network types, performing competitively with or outperforming state-of-the-art approaches such as the Leiden and Louvain Prune algorithmsespecially in large-scale networks.

The strength of FLMIG stems from its novel components, including an enhanced Louvain Prune procedure, which collectively improve the balance between exploration and exploitation during the search process. This leads to the recovery of more accurate

and well-connected communities. Furthermore, FLMIG requires minimal parameter tuning and employs an iterative refinement strategy that preserves community connectivityaddressing a common drawback in previous methods. Empirical results underscore the algorithms robustness, scalability, and overall effectiveness, establishing FLMIG as a practical and reliable tool for network analysis in various applications.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

The NP-hard challenge of community detection remains constrained by subjective definitions of communities and validation difficulties. Optimization-Based Methods (OBMs) have outperformed deterministic approaches. This work is motivated by an interest in developing novel optimization approaches to tackle this problem more effectively than current methods.

This thesis presents the Fast Local Move Iterated Greedy (FLMIG) algorithm, a new method for efficiently detecting communities in networks. FLMIG integrates fast local move heuristics into an iterated greedy framework to maximize modularity in a scalable and simple way. Experiments show it identifies high-quality communities, performing comparably or better than leading algorithms like Leiden and Louvain Prune, especially in large-scale networks.

FLMIG's performance is significantly boosted by its novel elements, such as the enhanced Louvain Prune algorithm. These features ensure a strong balance between exploring new solutions and refining existing ones, leading to the discovery of higher-quality communities. Furthermore, FLMIG requires little parameter tuning and uses an iterative process that maintains the internal connectivity of communities, overcoming a major flaw in previous methods. Computational results confirm the algorithm's robustness, scalability, and effectiveness, establishing it as a powerful tool for diverse network analysis applications.

We also present a new technique, the Clustering Coefficient Bat Algorithm (CC-DBAT), for detecting communities in complex networks. CC-DBAT uses the clustering coefficient to initialize its population. This population is then refined through a fast local

move and a random neighbor technique, which promotes solution diversity and enables rapid convergence toward the global optimum. Tests on both synthetic and real-world networks show that CC-DBAT is competitive against existing bat algorithm variants for this problem.

6.2 Future works

The path forward demands Hybrid Metaheuristics with Deep Structural Embeddings (HMCD-E) as a core paradigm. Embeddings transcend explicit topological metrics by distilling latent structural semantics, role-based relationships and multi-scale proximities, that modularity-driven methods (Louvain, MOCD) inherently miss. Early hybrids like SFCSA (node2vec + cuckoo search) and CD-GNN frameworks demonstrate this potential but lack systematic integration. Crucially, embeddings enable dynamic adaptability: techniques like NE2NMF and ECGNMF generate stable low-dimensional representations that support efficient incremental updates (e.g., DynaMos localized rules or δ-screening), unifying static and dynamic detection within HMCD-E architectures. Furthermore, embeddings provide semantically intelligent initialization, replacing stochastic starts in methods like MAGA-Net or CC-DBAT, to accelerate metaheuristic convergence.

HMCD-E directly resolves pervasive efficiency-accuracy trade-offs. Static accelerated heuristics (FLM, RNM) sacrifice solution quality for speed, while metaheuristics (LMOEA, MLACO) incur prohibitive computational costs. Dynamic consensus strategies (CCPSO, cSWO) suffer temporal drift, and localized updates (DynaMo) lack global coherence. Embeddings compress structural signatures to guide metaheuristics toward robust solutions while reducing computational overhead, simultaneously enhancing accuracy and scalability.

HMCD-E represents not an incremental step but a necessary leap. By fusing latent structural intelligence with adaptive optimization, it unlocks semantically grounded communities aligned with implicit network organization, real-time adaptability for evolving systems, and new frontiers in interpretable network sciencetranscending the limitations of current OBMs to meet next-generation analytical demands.

In addition, future work could explore extending FLMIG to detect overlapping communities and further reduce its computational time for extremely large datasets. The potential of parallel processing to decrease computation duration should also be investigated. Moreover, hybridizing FLMIG with network embedding techniques to handle temporal and multiplex networks, among other complex systems, could significantly expand its utility and influence in network science. This study not only contributes to the existing body of knowledge on community detection but also offers a practical and efficient tool for practitioners and researchers across multiple disciplines.

BIBLIOGRAPHY

[1] A. D. ABBOOD, B. A. ATTEA, A. A. HASAN, R. M. EVERSON, AND C. PIZZUTI, *Community detection model for dynamic networks based on hidden markov model and evolutionary algorithm*, Artificial Intelligence Review, 56 (2023), pp. 9665–9697.

[2] K. AGGARWAL AND A. ARORA, *Assessment of discrete bat-modified (dbat-m) optimization algorithm for community detection in complex network*, Arabian Journal for Science and Engineering, 48 (2023), pp. 2277–2296.

[3] U. ALON, *Network motifs: theory and experimental approaches*, Nature Reviews Genetics, 8 (2007), pp. 450–461.

[4] L. ANGELINI, S. BOCCALETTI, D. MARINAZZO, M. PELLICORO, AND S. STRAMAGLIA, *Identification of network modules by optimization of ratio association*, Chaos: An Interdisciplinary Journal of Nonlinear Science, 17 (2007).

[5] A. ARENAS, J. DUCH, A. FERNÁNDEZ, AND S. GÓMEZ, *Size reduction of complex networks preserving modularity*, New Journal of Physics, 9 (2007), p. 176.

[6] M. ASLAN AND İ. KOÇ, *Modified coot bird optimization algorithm for solving community detection problem in social networks*, Neural Computing and Applications, 36 (2024), pp. 5595–5619.

[7] M. AVAZBEIGI, *An overview of complexity theory*, Facility Location: Concepts, Models, Algorithms and Case Studies, (2009), pp. 19–36.

[8] D. A. BADER, H. MEYERHENKE, P. SANDERS, AND D. WAGNER, *Graph partitioning and graph clustering*, vol. 588, American Mathematical Society Providence, RI, 2013.

[9] S. BANSAL, S. BHOWMICK, AND P. PAYMAL, *Fast community detection for dynamic complex networks*, in Complex Networks: Second International Workshop, CompleNet 2010, Rio de Janeiro, Brazil, October 13-15, 2010, Revised Selected Papers, Springer, 2011, pp. 196–207.

[10] A.-L. BARABASI AND Z. N. OLTVAI, *Network biology: understanding the cell's functional organization*, Nature reviews genetics, 5 (2004), pp. 101–113.

[11] A. A. BARAA, A. D. ABBOOD, A. A. HASAN, C. PIZZUTI, M. AL-ANI, S. ÖZDEMIR, AND R. D. AL-DABBAGH, *A review of heuristics and metaheuristics for community detection in complex networks: Current usage, emerging development and future directions*, Swarm and Evolutionary Computation, 63 (2021), p. 100885.

[12] M. BAZARAA, H. SHERALI, AND C. SHETTY, *Nonlinear programming: Theory and algorithms*, Wiley-Interscience, 2006.

[13] R. K. BEHERA, D. NAIK, S. K. RATH, AND R. DHARAVATH, *Genetic algorithm-based community detection in large-scale social networks*, Neural Computing and Applications, 32 (2020), pp. 9649–9665.

[14] K. BERGERMANN AND M. STOLL, *Gradient flow-based modularity maximization for community detection in multiplex networks*, Philosophical Transactions A, 383 (2025), p. 20240244.

[15] F. BESHARATNIA, A. TALEBPOUR, AND S. ALIAKBARY, *An improved grey wolves optimization algorithm for dynamic community detection and data clustering*, Applied Artificial Intelligence, 36 (2022), p. 2012000.

[16] V. D. BLONDEL, J.-L. GUILLAUME, R. LAMBIOTTE, AND E. LEFEBVRE, *Fast unfolding of communities in large networks*, Journal of statistical mechanics: theory and experiment, 2008 (2008), p. P10008.

[17] C. BLUM AND A. ROLI, *Metaheuristics in combinatorial optimization: Overview and conceptual comparison*, ACM computing surveys (CSUR), 35 (2003), pp. 268–308.

[18] S. BOCCALETTI, G. BIANCONI, R. CRIADO, C. I. DEL GENIO, J. GÓMEZ-GARDENES, M. ROMANCE, I. SENDINA-NADAL, Z. WANG, AND M. ZANIN, *The structure and dynamics of multilayer networks*, Physics reports, 544 (2014), pp. 1–122.

[19] M. BOGUNÁ, R. PASTOR-SATORRAS, A. DÍAZ-GUILERA, AND A. ARENAS, *Models of social networks based on social distance attachment*, Physical review E, 70 (2004), p. 056122.

[20] ——, *Models of social networks based on social distance attachment*, Physical review E, 70 (2004), p. 056122.

[21] S. BOUAMAMA AND C. BLUM, *A randomized population-based iterated greedy algorithm for the minimum weight dominating set problem*, in 2015 6th international conference on information and communication systems (ICICS), IEEE, 2015, pp. 7–12.

[22] S. BOUAMAMA, C. BLUM, AND A. BOUKERRAM, *A population-based iterated greedy algorithm for the minimum weight vertex cover problem*, Applied Soft Computing, 12 (2012), pp. 1632–1639.

[23] S. BOUAMAMA, C. BLUM, AND P. PINACHO-DAVIDSON, *A population-based iterated greedy algorithm for maximizing sensor network lifetime*, Sensors, 22 (2022), p. 1804.

[24] F. BOUHATEM, A. A. EL HADJ, AND F. SOUAM, *Density-based approach with dual optimization for tracking community structure of increasing social networks*, International Journal on Artificial Intelligence Tools, 29 (2020), p. 2050002.

[25] R. BOUKABENE, F. BENBOUZID-SI TAYEB, AND N. DAKICHE, *Smart flattening cuckoo search algorithm for community detection in multiplex networks*, Social Network Analysis and Mining, 15 (2025), p. 39.

[26] U. BRANDES, D. DELLING, M. GAERTLER, R. GÖRKE, M. HOEFER, Z. NIKOLOSKI, AND D. WAGNER, *Maximizing modularity is hard*, arXiv preprint physics/0608255, (2006).

[27] U. BRANDES, D. DELLING, M. GAERTLER, R. GORKE, M. HOEFER, Z. NIKOLOSKI, AND D. WAGNER, *On modularity clustering*, IEEE transactions on knowledge and data engineering, 20 (2007), pp. 172–188.

[28] P. BRODKA AND T. GRECKI, *mlfr benchamark: Testing community detection algorithms in multi-layered*, Multiplex and Multiple Social Networks, (2014).

[29] P. BRÓDKA, S. SAGANOWSKI, AND P. KAZIENKO, *Ged: the method for group evolution discovery in social networks*, Social Network Analysis and Mining, 3 (2013), pp. 1–14.

[30] Q. CAI, M. GONG, B. SHEN, L. MA, AND L. JIAO, *Discrete particle swarm optimization for identifying community structures in signed social networks*, Neural Networks, 58 (2014), pp. 4–13.

[31] Q. CAI, L. MA, M. GONG, AND D. TIAN, *A survey on network community detection based on evolutionary computation*, International Journal of Bio-Inspired Computation, 8 (2016), pp. 84–98.

[32] A. CASADO, S. BERMUDO, A. LÓPEZ-SÁNCHEZ, AND J. SÁNCHEZ-ORO, *An iterated greedy algorithm for finding the minimum dominating set in graphs*, Mathematics and Computers in Simulation, 207 (2023), pp. 41–58.

[33] R. CAZABET, S. BOUDEBZA, AND G. ROSSETTI, *Evaluating community detection algorithms for progressively evolving graphs*, Journal of Complex Networks, 8 (2020), p. cnaa027.

[34] R. CAZABET, G. ROSSETTI, AND F. AMBLARD, *Dynamic community detection*, Encyclopedia of social network analysis and mining, (2017).

[35] F. CELLI, F. M. L. DI LASCIO, M. MAGNANI, B. PACELLI, AND L. ROSSI, *Social network data and practices: the case of friendfeed*, in International Conference on Social Computing, Behavioral Modeling, and Prediction, Springer, 2010, pp. 346–353.

[36] M. CHA, A. MISLOVE, AND K. P. GUMMADI, *A measurement-driven analysis of information propagation in the flickr social network*, in Proceedings of the 18th international conference on World wide web, 2009, pp. 721–730.

[37] F. CHENG, T. CUI, Y. SU, Y. NIU, AND X. ZHANG, *A local information based multi-objective evolutionary algorithm for community detection in complex networks*, Applied Soft Computing, 69 (2018), pp. 357–367.

[38] A. CLAUSET, M. E. NEWMAN, AND C. MOORE, *Finding community structure in very large networks*, Physical review E, 70 (2004), p. 066111.

[39] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN, *Introduction to algorithms*, MIT press, 2022.

[40] N. DAKICHE, F. BENBOUZID-SI TAYEB, AND K. BENATCHBA, *Transferring clique knowledge in multi-objective bee swarm optimization for dynamic community detection*, International Journal of Data Science and Analytics, (2025), pp. 1–22.

[41] N. DAKICHE, F. B.-S. TAYEB, Y. SLIMANI, AND K. BENATCHBA, *Tracking community evolution in social networks: A survey*, Information Processing & Management, 56 (2019), pp. 1084–1102.

[42] L. DANON, A. DIAZ-GUILERA, J. DUCH, AND A. ARENAS, *Comparing community structure identification*, Journal of statistical mechanics: Theory and experiment, 2005 (2005), p. P09008.

[43] S. DERRIBLE AND C. KENNEDY, *The complexity and robustness of metro networks*, Physica A: Statistical Mechanics and its Applications, 389 (2010), pp. 3678–3691.

[44] I. S. DHILLON, Y. GUAN, AND B. KULIS, *A unified view of kernel k-means, spectral clustering and graph cuts*, Citeseer, 2004.

[45] T. DOKEROGLU, E. SEVINC, T. KUCUKYILMAZ, AND A. COSAR, *A survey on new generation metaheuristic algorithms*, Computers & Industrial Engineering, 137 (2019), p. 106040.

[46] M. DORIGO, M. BIRATTARI, AND T. STUTZLE, *Ant colony optimization*, IEEE computational intelligence magazine, 1 (2007), pp. 28–39.

[47] J. DUCH AND A. ARENAS, *Community detection in complex networks using extremal optimization*, Physical review E, 72 (2005), p. 027104.

[48] N. DUHAN, A. SHARMA, AND K. K. BHATIA, *Page ranking algorithms: a survey*, in 2009 IEEE International Advance Computing Conference, IEEE, 2009, pp. 1530–1537.

[49] M. EHRGOTT, *Multicriteria optimization*, vol. 491, Springer Science & Business Media, 2005.

[50] T. EL-GHAZALI, *Metaheuristics: From design to implementation*, Wiley Publishing, 2009.

[51] T. A. FEO AND M. G. RESENDE, *Greedy randomized adaptive search procedures*, Journal of global optimization, 6 (1995), pp. 109–133.

[52] A. FERLIGOJ AND V. BATAGELJ, *Direct multicriteria clustering algorithms*, Journal of classification, 9 (1992), pp. 43–61.

[53] F. FOLINO AND C. PIZZUTI, *An evolutionary multiobjective approach for community discovery in dynamic networks*, IEEE Transactions on Knowledge and Data Engineering, 26 (2013), pp. 1838–1852.

[54] S. FORTUNATO, *Community detection in graphs*, Physics reports, 486 (2010), pp. 75–174.

[55] S. FORTUNATO AND D. HRIC, *Community detection in networks: A user guide*, Physics reports, 659 (2016), pp. 1–44.

[56] C. GAO, Z. CHEN, X. LI, Z. TIAN, S. LI, AND Z. WANG, *Multiobjective discrete particle swarm optimization for community detection in dynamic networks*, Europhysics Letters, 122 (2018), p. 28001.

[57] Z. GHALMANE, M. E. HASSOUNI, AND H. CHERIFI, *Immunization of networks with non-overlapping community structure*, Social Network Analysis and Mining, 9 (2019), pp. 1–22.

[58] N. GIRDHAR AND K. K. BHARADWAJ, *Community detection in signed social networks using multiobjective genetic algorithm*, Journal of the Association for Information Science and Technology, 70 (2019), pp. 788–804.

[59] M. GIRVAN AND M. E. NEWMAN, *Community structure in social and biological networks*, Proceedings of the national academy of sciences, 99 (2002), pp. 7821–7826.

[60] P. M. GLEISER AND L. DANON, *Community structure in jazz*, Advances in complex systems, 6 (2003), pp. 565–573.

[61] F. GLOVER, *Future paths for integer programming and links to artificial intelligence*, Computers & operations research, 13 (1986), pp. 533–549.

[62] F. GLOVER AND M. SAMORANI, *Intensification, diversification and learning in metaheuristic optimization*, Journal of Heuristics, 25 (2019), pp. 517–520.

[63] S. GÓMEZ, P. JENSEN, AND A. ARENAS, *Analysis of community structure in networks of correlated data*, Physical Review EStatistical, Nonlinear, and Soft Matter Physics, 80 (2009), p. 016114.

[64] M. GONG, L. MA, Q. ZHANG, AND L. JIAO, *Community detection in networks by using multiobjective evolutionary algorithm with decomposition*, Physica A: Statistical Mechanics and its Applications, 391 (2012), pp. 4050–4060.

[65] D. GREENE, D. DOYLE, AND P. CUNNINGHAM, *Tracking the evolution of communities in dynamic social networks*, in 2010 international conference on advances in social networks analysis and mining, IEEE, 2010, pp. 176–183.

[66] T. S. H. H. HOOS, *Stochastic Local Search: Foundations & Applications*, Elsevier / Morgan Kaufmann, San Francisco (CA), USA, 2005.

[67] L. HAMERS, Y. HEMERYCK, G. HERWEYERS, M. JANSSEN, H. KETERS, R. ROUSSEAU, AND A. VANHOUTTE, *Similarity measures in scientometric research: The jaccard index versus salton's cosine formula.*, Inf. Process. Manag., 25 (1989), pp. 315–318.

[68] E. A. HASSAN, A. I. HAFEZ, A. E. HASSANIEN, AND A. A. FAHMY, *A discrete bat algorithm for the community detection problem*, in Hybrid Artificial Intelligent Systems: 10th International Conference, HAIS 2015, Bilbao, Spain, June 22-24, 2015, Proceedings 10, Springer, 2015, pp. 188–199.

[69] P. HOLME AND J. SARAMÄKI, *Temporal networks*, Physics reports, 519 (2012), pp. 97–125.

[70] H. H. HOOS AND T. STÜTZLE, *Stochastic local search: Foundations & applications*, Elsevier / Morgan Kaufmann, San Francisco (CA), USA, 2004.

[71] HUMAN-COMPUTER INTERACTION LAB (HCIL), UNIVERSITY OF MARYLAND, *VAST challenge 2008*. `http://www.cs.umd.edu/hcil/VASTchallenge08/`, 2008. Accessed: 2025-08-04.

[72] P. R. HUNTER AND M. A. GASTON, *Numerical index of the discriminatory ability of typing systems: an application of simpson's index of diversity*, Journal of clinical microbiology, 26 (1988), pp. 2465–2466.

[73] Z. B. IMTIAZ, A. MANZOOR, S. UL ISLAM, M. A. JUDGE, K.-K. R. CHOO, AND J. J. RODRIGUES, *Discovering communities from disjoint complex networks using multi-layer ant colony optimization*, Future Generation Computer Systems, 115 (2021), pp. 659–670.

[74] R. INTERDONATO, A. TAGARELLI, D. IENCO, A. SALLABERRY, AND P. PONCELET, *Node-centric community detection in multilayer networks with layer-coverage diversification bias*, in Complex Networks VIII: Proceedings of the 8th Conference on Complex Networks CompleNet 2017 8, Springer, 2017, pp. 57–66.

[75] P. JIAO, W. YU, W. WANG, X. LI, AND Y. SUN, *Exploring temporal community structure and constant evolutionary pattern hiding in dynamic networks*, Neurocomputing, 314 (2018), pp. 224–233.

[76] D. E. JOSLIN AND D. P. CLEMENTS, *Squeaky wheel optimization*, Journal of Artificial Intelligence Research, 10 (1999), pp. 353–373.

[77] F. KARIMI, S. LOTFI, AND H. IZADKHAH, *Multiplex community detection in complex networks using an evolutionary approach*, Expert Systems with Applications, 146 (2020), p. 113184.

[78] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of computer computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.

[79] F. R. KHAWAJA, Z. ZHANG, Y. MEMON, AND A. ULLAH, *Exploring community detection methods and their diverse applications in complex networks: a comprehensive review*, Social Network Analysis and Mining, 14 (2024), p. 115.

[80] M. KIVELÄ, A. ARENAS, M. BARTHELEMY, J. P. GLEESON, Y. MORENO, AND M. A. PORTER, *Multilayer networks*, Journal of complex networks, 2 (2014), pp. 203–271.

[81] B. KLIMT AND Y. YANG, *The enron corpus: A new dataset for email classification research*, in European conference on machine learning, Springer, 2004, pp. 217–226.

[82] D. E. KNUTH, *The stanford graphbase: a platform for combinatorial algorithms*, in Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms, 1993, pp. 41–43.

[83] S. KOJAKU, F. RADICCHI, Y.-Y. AHN, AND S. FORTUNATO, *Network community detection via neural embeddings*, Nature Communications, 15 (2024), p. 9446.

[84] H. KONG, Q. KANG, W. LI, C. LIU, Y. KANG, AND H. HE, *A hybrid iterated carousel greedy algorithm for community detection in complex networks*, Physica A: Statistical Mechanics and its Applications, 536 (2019), p. 122124.

[85] B. KORTE AND J. VYGEN, *Combinatorial optimization: theory and algorithms*, Springer, 2008.

[86] R. KUMAR, J. NOVAK, AND A. TOMKINS, *Structure and evolution of online social networks*, in Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, 2006, pp. 611–617.

[87] S. KUMAR, A. MALLIK, AND S. S. SENGAR, *Community detection in complex networks using stacked autoencoders and crow search algorithm*, The Journal of Supercomputing, 79 (2023), pp. 3329–3356.

[88] J. KUNEGIS, *Konect: the koblenz network collection*, in Proceedings of the 22nd international conference on world wide web, 2013, pp. 1343–1350.

[89] A. LANCICHINETTI, S. FORTUNATO, AND J. KERTÉSZ, *Detecting the overlapping and hierarchical community structure in complex networks*, New journal of physics, 11 (2009), p. 033015.

[90] A. LANCICHINETTI, S. FORTUNATO, AND F. RADICCHI, *Benchmark graphs for testing community detection algorithms*, Physical Review EStatistical, Nonlinear, and Soft Matter Physics, 78 (2008), p. 046110.

[91] E. A. LEICHT AND M. E. NEWMAN, *Community structure in directed networks*, Physical review letters, 100 (2008), p. 118703.

[92] J. LESKOVEC AND R. SOSIČ, *Snap: A general-purpose network analysis and graph-mining library*, ACM Transactions on Intelligent Systems and Technology (TIST), 8 (2016), pp. 1–20.

[93] D. LI, X. ZHONG, Z. DOU, M. GONG, AND X. MA, *Detecting dynamic community by fusing network embedding and nonnegative matrix factorization*, Knowledge-Based Systems, 221 (2021), p. 106961.

[94] W. LI, Q. KANG, H. KONG, C. LIU, AND Y. KANG, *A novel iterated greedy algorithm for detecting communities in complex network*, Social Network Analysis and Mining, 10 (2020), pp. 1–17.

[95] Z. LI AND J. LIU, *A multi-agent genetic algorithm for community detection in complex networks*, Physica A: Statistical Mechanics and its Applications, 449 (2016), pp. 336–347.

[96] Z. LI, S. ZHANG, R.-S. WANG, X.-S. ZHANG, AND L. CHEN, *Quantitative function for community detection*, Physical Review EStatistical, Nonlinear, and Soft Matter Physics, 77 (2008), p. 036109.

[97] Y.-R. LIN, Y. CHI, S. ZHU, H. SUNDARAM, AND B. L. TSENG, *Facetnet: a framework for analyzing communities and their evolutions in dynamic networks*, in Proceedings of the 17th international conference on World Wide Web, 2008, pp. 685–694.

[98] C. LIU, Y. HAN, H. XU, S. YANG, K. WANG, AND Y. SU, *A community detection and graph-neural-network-based link prediction approach for scientific literature*, Mathematics, 12 (2024), p. 369.

[99] C. LIU, Q. KANG, H. KONG, W. LI, AND Y. KANG, *An iterated local search algorithm for community detection in complex networks*, International Journal of Modern Physics B, 34 (2020), p. 2050013.

[100] X. LIU AND T. MURATA, *Advanced modularity-specialized label propagation algorithm for detecting communities in networks*, Physica A: Statistical Mechanics and its Applications, 389 (2010), pp. 1493–1500.

[101] C. W. LOE AND H. J. JENSEN, *Comparison of communities detection algorithms for multiplex*, Physica A: Statistical Mechanics and its Applications, 431 (2015), pp. 29–45.

[102] M. LOZANO AND F. J. RODRÍGUEZ, *Iterated greedy*, in Discrete Diversity and Dispersion Maximization: A Tutorial on Metaheuristic Optimization, Springer, 2023, pp. 107–133.

[103] D. LUSSEAU, K. SCHNEIDER, O. J. BOISSEAU, P. HAASE, E. SLOOTEN, AND S. 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 (2003), pp. 396–405.

[104] X. MA AND D. DONG, *Evolutionary nonnegative matrix factorization algorithms for community detection in dynamic networks*, IEEE transactions on knowledge and data engineering, 29 (2017), pp. 1045–1058.

[105] M. MADADYAR, V. MAJIDNEZHAD, J. BAGHERZADEH MOHASEFI, AND F. SOLEIMANIAN GHAREHCHOPOGH, *An enhanced hybrid optimization algorithm for community detection in social network*, International Journal of Data Science and Analytics, (2025), pp. 1–23.

[106] M. MAGNANI, O. HANTEER, R. INTERDONATO, L. ROSSI, AND A. TAGARELLI, *Community detection in multiplex networks*, ACM Computing Surveys (CSUR), 54 (2021), pp. 1–35.

[107] M. MAZZA, G. COLA, AND M. TESCONI, *Modularity-based approach for tracking communities in dynamic social networks*, Knowledge-Based Systems, 281 (2023), p. 111067.

[108] I. MESSAOUDI AND N. KAMEL, *A multi-objective bat algorithm for community detection on dynamic social networks*, Applied Intelligence, 49 (2019), pp. 2119–2136.

[109] A. MISLOVE, H. S. KOPPULA, K. P. GUMMADI, P. DRUSCHEL, AND B. BHATTACHARJEE, *Growth of the flickr social network*, in Proceedings of the first workshop on Online social networks, 2008, pp. 25–30.

[110] A. MISLOVE, M. MARCON, K. P. GUMMADI, P. DRUSCHEL, AND B. BHATTACHARJEE, *Measurement and analysis of online social networks*, in Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, 2007, pp. 29–42.

[111] D. MOLINA, J. POYATOS, J. D. SER, S. GARCÍA, A. HUSSAIN, AND F. HERRERA, *Comprehensive taxonomies of nature-and bio-inspired optimization: Inspiration versus algorithmic behavior, critical analysis recommendations*, Cognitive Computation, 12 (2020), pp. 897–939.

[112] P. J. MUCHA, T. RICHARDSON, K. MACON, M. A. PORTER, AND J.-P. ONNELA, *Community structure in time-dependent, multiscale, and multiplex networks*, science, 328 (2010), pp. 876–878.

[113] R. Y. NAKAMURA, L. A. PEREIRA, K. A. COSTA, D. RODRIGUES, J. P. PAPA, AND X.-S. YANG, *Bba: a binary bat algorithm for feature selection*, in 2012 25th SIBGRAPI conference on graphics, patterns and images, IEEE, 2012, pp. 291–297.

[114] M. E. NEWMAN, *Fast algorithm for detecting community structure in networks*, Physical review E, 69 (2004), p. 066133.

[115] ——, *Finding community structure in networks using the eigenvectors of matrices*, Physical review E, 74 (2006), p. 036104.

[116] M. E. NEWMAN AND M. GIRVAN, *Finding and evaluating community structure in networks*, Physical review E, 69 (2004), p. 026113.

[117] M. E. J. NEWMAN, *Network datasets*. `http://www-personal.umich.edu/~mejn/netdata/`, 2013. Accessed on 07/03/2021.

[118] N. P. NGUYEN, T. N. DINH, Y. SHEN, AND M. T. THAI, *Dynamic social community detection and its applications*, PloS one, 9 (2014), p. e91431.

[119] V. NICOSIA, G. MANGIONI, V. CARCHIOLO, AND M. MALGERI, *Extending the definition of modularity to directed graphs with overlapping communities*, Journal of Statistical Mechanics: Theory and Experiment, 2009 (2009), p. P03024.

[120] J.-P. ONNELA, J. SARAMÄKI, J. HYVÖNEN, G. SZABÓ, D. LAZER, K. KASKI, J. KERTÉSZ, AND A.-L. BARABÁSI, *Structure and tie strengths in mobile communication networks*, Proceedings of the national academy of sciences, 104 (2007), pp. 7332–7336.

[121] E. OSABA, J. DEL SER, D. CAMACHO, M. N. BILBAO, AND X.-S. YANG, *Community detection in networks using bio-inspired optimization: Latest developments, new results and perspectives with a selection of recent meta-heuristics*, Applied Soft Computing, 87 (2020), p. 106010.

[122] N. OZAKI, H. TEZUKA, AND M. INABA, *A simple acceleration method for the louvain algorithm*, International Journal of Computer and Electrical Engineering, 8 (2016), p. 207.

[123] G. PALLA, A.-L. BARABÁSI, AND T. VICSEK, *Quantifying social group evolution*, Nature, 446 (2007), pp. 664–667.

[124] ——, *Quantifying social group evolution*, Nature, 446 (2007), pp. 664–667.

[125] Y. PAN, D.-H. LI, J.-G. LIU, AND J.-Z. LIANG, *Detecting community structure in complex networks via node similarity*, Physica A: Statistical Mechanics and its Applications, 389 (2010), pp. 2849–2857.

[126] S. PAUL, C. KONER, A. MITRA, AND S. GHOSH, *A study on algorithms for detection of communities in dynamic social networks: A review*, in International Conference on Computational Intelligence in Communications and Business Analytics, Springer, 2023, pp. 51–64.

[127] L. PEEL, D. B. LARREMORE, AND A. CLAUSET, *The ground truth about metadata and community detection in networks*, Science advances, 3 (2017), p. e1602548.

[128] C. PIZZUTI, *Ga-net: A genetic algorithm for community detection in social networks*, in International conference on parallel problem solving from nature, Springer, 2008, pp. 1081–1090.

[129] ——, *Evolutionary computation for community detection in networks: A review*, IEEE Transactions on Evolutionary Computation, 22 (2017), pp. 464–483.

[130] F. RADICCHI, C. CASTELLANO, F. CECCONI, V. LORETO, AND D. PARISI, *Defining and identifying communities in networks*, Proceedings of the national academy of sciences, 101 (2004), pp. 2658–2663.

[131] K. RAJWAR, K. DEEP, AND S. DAS, *An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges*, Artificial Intelligence Review, 56 (2023), pp. 13187–13257.

[132] S. RANJKESH, B. MASOUMI, AND S. M. HASHEMI, *A novel robust memetic algorithm for dynamic community structures detection in complex networks*, World Wide Web, 27 (2024), p. 3.

[133] J. REICHARDT AND S. BORNHOLDT, *Statistical mechanics of community detection*, Phys. Rev. E, 74 (2006), p. 016110.

[134] J. REICHARDT AND S. BORNHOLDT, *Statistical mechanics of community detection*, Physical Review EStatistical, Nonlinear, and Soft Matter Physics, 74 (2006), p. 016110.

[135] A. REIHANIAN, M.-R. FEIZI-DERAKHSHI, AND H. S. AGHDASI, *An enhanced multi-objective biogeography-based optimization for overlapping community detection in social networks with node attributes*, Information Sciences, 622 (2023), pp. 903–929.

[136] S. ROMANO, J. BAILEY, V. NGUYEN, AND K. VERSPOOR, *Standardized mutual information for clustering comparisons: one step further in adjustment for chance*, in International conference on machine learning, PMLR, 2014, pp. 1143–1151.

[137] G. ROSSETTI, *: graph benchmark handling community dynamics*, Journal of Complex Networks, 5 (2017), pp. 893–912.

[138] G. ROSSETTI AND R. CAZABET, *Community discovery in dynamic networks: a survey*, ACM computing surveys (CSUR), 51 (2018), pp. 1–37.

[139] G. ROSSETTI, L. PAPPALARDO, D. PEDRESCHI, AND F. GIANNOTTI, *Tiles: an online algorithm for community discovery in dynamic social networks*, Machine Learning, 106 (2017), pp. 1213–1241.

[140] G. ROSSETTI, L. PAPPALARDO, AND S. RINZIVILLO, *A novel approach to evaluate community detection algorithms on ground truth*, in Complex Networks VII: Proceedings of the 7th Workshop on Complex Networks CompleNet 2016, Springer, 2016, pp. 133–144.

[141] L. ROSSI AND M. MAGNANI, *Towards effective visual analytics on multiplex and multilayer networks*, Chaos, Solitons & Fractals, 72 (2015), pp. 68–76.

[142] F. ROTHLAUF ET AL., *Design of modern heuristics: principles and application*, vol. 8, Springer, 2011.

[143] R. RUIZ AND T. STÜTZLE, *A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem*, European journal of operational research, 177 (2007), pp. 2033–2049.

[144] A. SAID, R. A. ABBASI, O. MAQBOOL, A. DAUD, AND N. R. ALJOHANI, *Cc-ga: A clustering coefficient based genetic algorithm for detecting communities in social networks*, Applied Soft Computing, 63 (2018), pp. 59–70.

[145] M. SALATHÉ AND J. H. JONES, *Dynamics and control of diseases in networks with community structure*, PLoS computational biology, 6 (2010), p. e1000736.

[146] B. SALIM, *Design Of A Learning Method For Automatic Data Extraction*, PhD thesis, Université Ferhat Abbas - Sétif 1, 2014.

[147] J. SANCHEZ-ORO AND A. DUARTE, *Iterated greedy algorithm for performing community detection in social networks*, Future Generation Computer Systems, 88 (2018), pp. 785–791.

[148] J. SHANG, L. LIU, F. XIE, Z. CHEN, J. MIAO, X. FANG, AND C. WU, *A real-time detecting algorithm for tracking community structure of dynamic networks*, arXiv preprint arXiv:1407.2683, (2014).

[149] H. SHEN, X. CHENG, K. CAI, AND M.-B. HU, *Detect overlapping and hierarchical community structure in networks*, Physica A: Statistical Mechanics and its Applications, 388 (2009), pp. 1706–1712.

[150] C. SHI, Q. CAI, G. WANG, AND F. JIANG, *A new genetic algorithm for community detection*, in Complex Sciences, First International Conference, Complex 2009, Shanghai, China, February 23-25, 2009, Revised Papers, Part II, vol. 5592 of Lecture Notes in Computer Science, Springer, 2009, pp. 110–121.

[151] C. SHI, Z. YAN, Y. CAI, AND B. WU, *Multi-objective community detection in complex networks*, Applied Soft Computing, 12 (2012), pp. 850–859.

[152] C. SHI, P. S. YU, Y. CAI, Z. YAN, AND B. WU, *On selection of objective functions in multi-objective community detection*, in Proceedings of the 20th ACM international conference on Information and knowledge management, 2011, pp. 2301–2304.

[153] J. SHI AND J. MALIK, *Normalized cuts and image segmentation*, IEEE Transactions on pattern analysis and machine intelligence, 22 (2000), pp. 888–905.

[154] S. T. SHISHAVAN AND F. S. GHAREHCHOPOGH, *An improved cuckoo search optimization algorithm with genetic algorithm for community detection in complex networks*, Multimedia Tools and Applications, 81 (2022), pp. 25205–25231.

[155] E. A. SILVER, *An overview of heuristic solution methods*, Journal of the operational research society, 55 (2004), pp. 936–956.

[156] A. SONG, M. LI, X. DING, W. CAO, AND K. PU, *Community detection using discrete bat algorithm*, IAENG International Journal of Computer Science, 43 (2016), pp. 37–43.

[157] A. G. SPAMPINATO, R. A. SCOLLO, V. CUTELLO, AND M. PAVONE, *Random search immune algorithm for community detection*, Soft Computing, 27 (2023), pp. 8061–8090.

[158] T. STÜTZLE, *Iterated local search for the quadratic assignment problem*, European journal of operational research, 174 (2006), pp. 1519–1539.

[159] S. TAIBI, S. BOUAMAMA, AND L. TOUMI, *An enhanced bat algorithm using clustering coefficient for community detection in complex networks*, in 2024 1st International Conference on Innovative and Intelligent Information Technologies (IC3IT), IEEE, 2024, pp. 1–6.

[160] S. TAIBI, L. TOUMI, AND S. BOUAMAMA, *Complex network community discovery using fast local move iterated greedy algorithm*, The Journal of Supercomputing, 81 (2025), p. 182.

[161] E.-G. TALBI, *A taxonomy of hybrid metaheuristics*, Journal of heuristics, 8 (2002), pp. 541–564.

[162] V. A. TRAAG, *Faster unfolding of communities: Speeding up the louvain algorithm*, Physical Review E, 92 (2015), p. 032801.

[163] V. A. TRAAG, L. WALTMAN, AND N. J. VAN ECK, *From louvain to leiden: guaranteeing well-connected communities*, Scientific reports, 9 (2019), p. 5233.

[164] P. J. VAN LAARHOVEN, E. H. AARTS, P. J. VAN LAARHOVEN, AND E. H. AARTS, *Simulated annealing*, Springer, 1987.

[165] B. VISWANATH, A. MISLOVE, M. CHA, AND K. P. GUMMADI, *On the evolution of user interaction in facebook*, in Proceedings of the 2nd ACM workshop on Online social networks, 2009, pp. 37–42.

[166] L. WALTMAN AND N. J. VAN ECK, *A smart local moving algorithm for large-scale modularity-based community detection*, The European physical journal B, 86 (2013), pp. 1–14.

[167] Z. WANG, C. WANG, C. GAO, X. LI, AND X. LI, *An evolutionary autoencoder for dynamic community detection*, Science China Information Sciences, 63 (2020), p. 212205.

[168] T. WASKIEWICZ, *Friend of a friend influence in terrorist social networks*, in Proceedings on the international conference on artificial intelligence (ICAI), The Steering Committee of The World Congress in Computer Science, Computer , 2012, p. 1.

[169] D. J. WATTS AND S. H. STROGATZ, *Collective dynamics of 'small-world' networks*, nature, 393 (1998), pp. 440–442.

[170] ——, *Collective dynamics of small-worldnetworks*, nature, 393 (1998), pp. 440–442.

[171] Y.-C. WEI AND C.-K. CHENG, *Ratio cut partitioning for hierarchical designs*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 10 (1991), pp. 911–921.

[172] J. YANG AND J. LESKOVEC, *Defining and evaluating network communities based on ground-truth*, in Proceedings of the ACM SIGKDD workshop on mining data semantics, 2012, pp. 1–8.

[173] X.-S. YANG, *Engineering optimization: an introduction with metaheuristic applications*, John Wiley & Sons, 2010.

[174] ——, *A new metaheuristic bat-inspired algorithm*, in Nature inspired cooperative strategies for optimization (NICSO 2010), Springer, 2010, pp. 65–74.

[175] Y. YIN, Y. ZHAO, H. LI, AND X. DONG, *Multi-objective evolutionary clustering for large-scale dynamic community detection*, Information Sciences, 549 (2021), pp. 269–287.

[176] L. YU, X. ZHAO, M. LV, AND J. ZHANG, *A consensus community-based spider wasp optimization for dynamic community detection*, Mathematics, 13 (2025), p. 265.

[177] W. YU, W. WANG, P. JIAO, AND X. LI, *Evolutionary clustering via graph regularized nonnegative matrix factorization for exploring temporal networks*, Knowledge-Based Systems, 167 (2019), pp. 1–10.

[178] W. W. ZACHARY, *An information flow model for conflict and fission in small groups*, Journal of anthropological research, 33 (1977), pp. 452–473.

[179] N. ZARAYENEH AND A. KALYANARAMAN, *Delta-screening: a fast and efficient technique to update communities in dynamic graphs*, IEEE transactions on network science and engineering, 8 (2021), pp. 1614–1629.

[180] X. ZENG, W. WANG, C. CHEN, AND G. G. YEN, *A consensus community-based particle swarm optimization for dynamic community detection*, IEEE transactions on cybernetics, 50 (2019), pp. 2502–2513.

[181] K. ZHANG, Z. ZHANG, K. BIAN, J. XU, AND J. GAO, *A personalized next-song recommendation system using community detection and markov model*, in 2017 IEEE Second International Conference on Data Science in Cyberspace (DSC), IEEE, 2017, pp. 118–123.

[182] W. ZHANG, S. YU, L. WANG, W. GUO, AND M.-F. LEUNG, *Constrained symmetric non-negative matrix factorization with deep autoencoders for community detection*, Mathematics, 12 (2024), p. 1554.

[183] W. ZHANG, R. ZHANG, R. SHANG, J. LI, AND L. JIAO, *Application of natural computation inspired method in community detection*, Physica A: Statistical Mechanics and its Applications, 515 (2019), pp. 130–150.

[184] Y. ZHANG, Y. LIU, J. LI, J. ZHU, C. YANG, W. YANG, AND C. WEN, *Wocda: A whale optimization based community detection algorithm*, Physica A: Statistical Mechanics and its Applications, 539 (2020), p. 122937.

[185] D. ZHUANG, J. M. CHANG, AND M. LI, *Dynamo: Dynamic community detection by incrementally maximizing modularity*, IEEE Transactions on Knowledge and Data Engineering, 33 (2019), pp. 1934–1945.