A High-Performance Evolutionary Multiobjective Community Detection Algorithm

Guilherme O. Santos[1*], Lucas S. Vieira[1], Giulio Rossetti[3], Carlos H. G. Ferreira[1], Gladston J. P. Moreira[2]

[1*]Departamento de Computação e Sistemas, Universidade Federal de Ouro Preto, João Monlevade, 35931-008, Minas Gerais, Brazil.
[2]Computing Department, Universidade Federal de Ouro Preto, Ouro Preto, 35402-136, Minas Gerais, Brazil.
[3]Istituto di Scienza e Tecnologie dell'Informazione, National Research Council of Italy, Pisa, 56127, PI, Italy.

*Corresponding author(s). E-mail(s): guilherme.os1@aluno.ufop.edu.br;
Contributing authors: lucas.sv@aluno.ufop.edu.br;
giulio.rossetti@isti.cnr.it; chgferreira@ufop.edu.br; gladston@ufop.edu.br ;

Abstract

Community structure is a key feature of complex networks, underpinning diverse phenomena across social, biological, and technological systems. While traditional methods like Louvain and Leiden offer efficient solutions, they rely on single-objective optimization, often failing to capture the multifaceted nature of real-world networks. Multi-objective approaches address this limitation by considering multiple structural criteria simultaneously, but their high computational cost restricts their use in large-scale settings. We propose HP-MOCD, a high-performance, fully parallel evolutionary algorithm based on NSGA-II, designed to uncover high-quality community structures by jointly optimizing conflicting objectives. HP-MOCD leverages topology-aware genetic operators and parallelism to efficiently explore the solution space and generate a diverse Pareto front of community partitions. Experimental results on large synthetic benchmarks demonstrate that HP-MOCD consistently outperforms existing multi-objective methods in runtime, while achieving superior or comparable detection accuracy. These findings position HP-MOCD as a scalable and practical solution for community detection in large, complex networks.

Keywords: community detection, complex networks, multi-objective optimization, evolutionary algorithms, NSGA-II, parallel computing

1 Introduction

The study of complex systems, ubiquitous in social, technological, and biological domains, heavily relies on modeling their underlying connectivity patterns as complex networks (Boccaletti et al. 2006). Examples range from social media platforms and the World Wide Web to protein-protein interaction networks and critical infrastructures (Ravasz et al. 2002). A key characteristic of many such networks is their organization into communities, groups of nodes with dense internal connections and sparser connections to the rest of the network (Newman 2006). Identifying these communities is crucial for understanding network organization, function, and dynamics, making community detection a central research area in network science (Fortunato and Hric 2016; Flake et al. 2002; Blondel et al. 2008; Shi et al. 2010). Single-objective methods, while prevalent, often oversimplify the inherent complexity of real-world network structures, failing to capture their true multifaceted nature.

The task of community detection is commonly formulated as a combinatorial optimization problem, where the goal is to partition the network into clusters that maximize internal cohesion while minimizing interconnections between groups (Newman and Girvan 2004). Since this problem is NP-hard, most practical solutions rely on traditional heuristic methods that optimize a predefined structural quality metric. In this work, we adopt a crisp partitioning model, where each node is assigned to exactly one community, reflecting a hard clustering approach that assumes mutually exclusive group membership. Among the most widely adopted are the Louvain algorithm (Blondel et al. 2008), which greedily maximizes modularity (Q), and its improved variant, Leiden (Traag et al. 2019), which addresses issues such as disconnected communities through an enhanced refinement process.

While effective and computationally efficient, these traditional heuristic methods exhibit notable limitations. Their reliance on a single metric like modularity introduces structural biases, including the well-documented resolution limit, which causes small but meaningful communities to be absorbed into larger ones (Fortunato and Barthélemy 2007; Newman and Girvan 2004). Modularity-based methods may also suffer from degeneracy, producing multiple partitions with similar scores but different topologies (Traag et al. 2019). Although Leiden mitigates some of these issues by ensuring better-connected communities and more stable partitions, it still operates under a fixed optimization criterion. This inherently restricts their capacity to navigate the complex trade-offs that arise in real-world networks, where no single metric fully captures the richness of the underlying structure (Azevedo et al. 2024).

Recognizing the limitations of traditional heuristic methods, the field has progressively shifted towards multi-objective evolutionary algorithms (MOEAs) as a more expressive and flexible framework for community detection. Unlike single-metric approaches, MOEAs aim to optimize multiple, often conflicting, objectives simultaneously. This multi-criteria formulation enables a more nuanced characterization of community structures by capturing diverse structural features of the network, such as internal density, inter-community sparsity, and modularity. By design, these algorithms produce a Pareto front, i.e., a set of non-dominated solutions, offering practitioners a range of alternatives rather than a single fixed partition (Moreira and Paquete

2019). This enables a more comprehensive exploration of the solution space, facilitating informed decisions based on application-specific trade-offs.

Early works such as MOGA-Net (Pizzuti 2009), which jointly optimized a community score and a fitness-based criterion, and Shi-MOCD (Shi et al. 2010), which explicitly targeted intra-community density and inter-community sparsity, demonstrated the potential of MOEAs to overcome structural limitations of modularity-based methods, including the resolution limit (Fortunato and Barthélemy 2007). Other metaheuristics, including adaptations of the Bat Algorithm (Hussein et al. 2015), further expanded the design space by exploring alternative population-based strategies. Despite their theoretical appeal, the practical adoption of MOEAs for large-scale community detection remains limited. A key challenge is their computational cost. Maintaining and evolving a population of candidate solutions requires repeated dominance comparisons, Pareto sorting, and fitness evaluations, all of which can be expensive in networks with thousands or millions of nodes and edges. Many existing implementations exhibit time complexities on the order of $O(n^2)$ or worse, primarily due to operations involving pairwise comparisons and graph traversals (Shi et al. 2010; Pizzuti 2012). This makes them impractical for real-time or large-scale applications without specialized acceleration strategies.

Additionally, few high-performance implementations of MOEAs are publicly available for community detection, which limits accessibility for researchers and practitioners. Most available tools are either experimental, tied to specific datasets, or lack parallelization and optimization for modern hardware. Finally, designing effective genetic operators that preserve meaningful community structures while ensuring exploration of the vast combinatorial search space remains an open problem. Balancing exploitation and exploration in this context is especially difficult due to the discrete and topologically constrained nature of the community detection problem.

To address these challenges, this paper introduces the High-Performance Multi-Objective Community Detection (HP-MOCD) algorithm. HP-MOCD is a scalable evolutionary algorithm based on the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) framework (Deb et al. 2002), specifically engineered for accuracy and efficiency in large complex networks. It employs a parallel computing architecture and custom genetic operators that incorporate topological information, inspired by Shi et al. (2010), to effectively explore the solution space. Instead of yielding a single partition, HP-MOCD produces a diverse set of Pareto-optimal solutions, each representing a different trade-off between conflicting structural objectives, thereby offering a more holistic understanding of network community structures. Moreover, this solution set can be flexibly analyzed under multiple evaluation metrics or selected according to the specific needs of a given application. Our primary contributions are as follows:

(i) We introduce novel crossover and mutation schemes that both refine community assignments and maintain population diversity, leading to higher-quality partitions.
(ii) By embedding Pareto-front selection within the NSGA-II framework, we jointly optimize competing structural criteria, yielding a rich solution set adaptable to multiple analytical needs.

(iii) We prove that, for the sparse graphs typically encountered in practice ($|E| = O(|V|)$) with fixed population size N_p, the per-generation cost reduces to $O\big(N_p\,|V|\big)$, and our experiments confirm runtimes remain well below worst-case bounds.
(iv) We design a fully multi-threaded architecture that exploits modern multi-core processors, drastically reducing runtime on large-scale networks.
(v) We provide a Free/Libre and Open Source Software (FLOSS) implementation of HP-MOCD, enabling researchers and practitioners to apply, extend, and benchmark our approach.

The remainder of this paper is structured as follows. Section 2 reviews foundational concepts and prior work in community detection, covering both traditional heuristics and multi-objective evolutionary approaches. Section 3 introduces the HP-MOCD algorithm in detail, outlining its architecture, genetic operators, and optimization strategy. Section 4 describes the experimental setup and presents a comparative evaluation against established methods. Finally, Section 5 discusses the results, highlights practical implications, and suggests directions for future research.

2 Fundamentals and Related Work

Community detection is a fundamental problem in network science, central to understanding the structure and function of complex systems. This section first introduces the foundational definitions and modeling assumptions adopted in this work. We then briefly review traditional heuristic methods, with emphasis on modularity-based algorithms such as Louvain and Leiden, analyzing their performance and limitations. Finally, we present an overview of multi-objective community detection, discussing its theoretical foundations, algorithmic strategies, and the key contributions from the literature that inform the design of our proposed method.

2.1 The definition of community

Community structure is a central concept in the analysis of complex networks, referring to the tendency of nodes to organize into groups with dense internal connections and relatively sparse connections to the rest of the network. While no universally accepted definition exists, one of the most common structural formulations is based on the relative density of intra- and inter-community links (Radicchi et al. 2004). In this perspective, communities are understood as subgraphs in which internal connectivity is significantly stronger than external connectivity.

Several formulations of community detection have emerged in the literature, depending on the nature of the data and the intended application (Fortunato 2010). Among the most common is node clustering, where the goal is to assign each node to a community. Other approaches include edge clustering, which focuses on grouping edges rather than nodes, and overlapping or fuzzy clustering, in which nodes may belong to multiple communities, either to the same degree or with varying levels of association. More specialized formulations incorporate additional structural or semantic information (Lancichinetti and Fortunato 2009). Attributed clustering, for example,

leverages both network topology and node-level features, while bipartite clustering targets networks composed of two distinct node types. Temporal clustering extends the task to dynamic networks, capturing how communities evolve over time. In directed acyclic graphs (DAGs), antichain clustering is used to identify groups based on partial order relationships (Rossetti et al. 2019).

In this work, we focus on node clustering under a crisp partitioning model, also referred to as hard clustering, where each node is assigned to exactly one community. This assumption of mutually exclusive group membership simplifies the structural analysis and remains the most widely adopted formulation in the evaluation of community detection algorithms. We further restrict our scope to undirected and unweighted networks, which are commonly used in benchmarks and allow clear structural interpretations based solely on topology.

Formally, let $G(V, E)$ be an undirected and unweighted graph, where V is the set of nodes and E the set of edges. The goal of community detection is to find a partition $C = \{C_1, C_2, \ldots, C_k\}$ of the vertex set such that:

$$\bigcup_{i=1}^{k} C_i = V \quad \text{and} \quad C_i \cap C_j = \emptyset \quad \forall\, i \neq j \tag{1}$$

In the remainder of this section, we discuss the main algorithmic approaches developed to tackle this problem. We begin with traditional heuristic methods, which have dominated the field due to their efficiency and simplicity. We then turn to more recent efforts based on multi-objective optimization, which more closely align with the direction pursued in this work.

2.2 Traditional algorithms

Traditional approaches to community detection in complex networks often rely on the optimization of a structural quality function, the most prominent being modularity. These methods aim to partition a network such that the density of edges within communities is significantly higher than between them. While modularity-based algorithms remain dominant due to their efficiency and simplicity, several alternative heuristics have also shaped the field. Among the earliest and most influential is the Girvan–Newman algorithm, which detects communities by iteratively removing edges with the highest betweenness centrality, gradually decomposing the network into disconnected components (Girvan and Newman 2002). Despite its conceptual elegance, this method suffers from high computational cost and limited scalability.

Another prominent strategy is label propagation, a fast heuristic where each node adopts the most frequent label among its neighbors. This process converges to a partition where nodes in the same community share the same label (Raghavan et al. 2007). Label propagation is highly scalable but can yield unstable results and lacks a clear objective function, making its outcomes harder to interpret. Among modularity-based methods, the Louvain algorithm proposed by Blondel et al. (2008) has become one of the most widely used due to its balance between accuracy and computational performance. It seeks to optimize the modularity metric, defined for undirected and

unweighted networks as:

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

In this equation, A_{ij} represents the adjacency matrix of the graph, k_i and k_j denote the degrees of nodes i and j, m is the total number of edges, and $\delta(c_i, c_j)$ is an indicator function equal to 1 when nodes i and j belong to the same community and 0 otherwise. The Louvain method operates in two main phases. Initially, each node is placed in its own community, and the algorithm iteratively moves nodes to neighboring communities if doing so increases modularity. Once no further local improvement is possible, communities are aggregated into super-nodes, and the process is repeated on the resulting coarsened graph. This continues until convergence.

Despite its popularity and efficiency, the Louvain algorithm exhibits several well-known limitations that affect both the quality and stability of the resulting partitions (Fortunato and Barthélemy 2007). Some works have attempted to address these limitations (Wang et al. 2019). One key issue is the production of disconnected communities, a consequence of how nodes are greedily reassigned based solely on local modularity gain without enforcing internal connectivity constraints (Fortunato and Hric 2016). This can lead to communities that are not internally cohesive, violating a fundamental structural assumption in community detection. Furthermore, Louvain is sensitive to the order in which nodes are processed during the local moving phase. Because the algorithm applies greedy updates, different node orderings may lead to different partitions with comparable modularity scores, introducing a level of non-determinism that compromises reproducibility.

A more fundamental limitation lies in the modularity function itself, particularly its susceptibility to the resolution limit problem. As demonstrated by Fortunato and Barthélemy (2007), modularity optimization tends to favor large communities and may fail to identify smaller but structurally meaningful groups, especially in networks with heterogeneous community sizes. This limitation arises because modularity evaluates gains relative to a global null model, which biases the optimization toward partitions that absorb smaller communities into larger ones. Additionally, the modularity landscape often contains many local maxima, leading to a high degree of degeneracy in the solution space. This means that multiple partitions with very different community structures can yield similar modularity values, complicating the interpretation and comparison of results across runs.

To address some of these issues, the Leiden algorithm was introduced by Traag et al. (2019). It preserves the modularity optimization framework but improves upon Louvain by enforcing connectedness within communities and refining the aggregation process. The algorithm proceeds in three stages. The first is local moving, in which nodes are reassigned to neighboring communities to improve a quality function. The second is refinement, which ensures that all communities remain internally connected, splitting any that are not. The final stage is aggregation, where communities are collapsed into single nodes, and the process restarts on the simplified graph.

While the Leiden algorithm was originally introduced with support for multiple objective functions, it often employs the Constant Potts Model (CPM) instead of modularity. CPM defines community quality in terms of internal edge density, controlled by a resolution parameter γ, and is given by:

$$H = \sum_c \left(e_c - \gamma \frac{n_c(n_c - 1)}{2} \right) \tag{3}$$

In this expression, e_c denotes the number of internal edges in community c, n_c is the number of nodes in that community, and γ sets the minimum internal density threshold. By tuning γ, the algorithm can control the granularity of the resulting partition, identifying larger or smaller communities accordingly. Unlike modularity, CPM does not suffer from the same theoretical resolution limit because it is a local objective function, and its optimization does not depend on global graph properties such as the total number of edges.

Nevertheless, in practical applications, CPM introduces its own set of challenges. The performance of the algorithm can vary significantly with the choice of γ, which often requires careful, problem-specific calibration. Furthermore, when Leiden is configured to use modularity as its objective function, as is commonly done for compatibility with existing benchmarks, it inherits many of the same limitations as Louvain, including the resolution limit and modularity degeneracy. In either case, whether using modularity or CPM, Leiden remains fundamentally a single-objective optimization method.

Both Louvain and Leiden represent the current state of practice in scalable, modularity-based community detection. While they offer practical advantages and have become fact standard in many applications, their reliance on a single optimization criterion limits their expressiveness. In particular, they are not designed to capture multiple, potentially conflicting structural properties simultaneously. These limitations motivate the investigation of multi-objective optimization strategies, which we explore in the next section.

2.3 Multi-Objective algorithms

Traditional community detection algorithms are typically designed to optimize a single objective function, such as modularity. While efficient and widely used, these approaches exhibit critical limitations. They often produce narrow solutions that emphasize specific structural properties at the expense of others, show inconsistent performance across different network topologies, and depend heavily on the choice of objective function. These factors limit their ability to fully capture the multifaceted nature of community structure.

To overcome these shortcomings, a multi-objective optimization approach provides a more expressive and flexible framework. A multi-objective optimization problem (MOP) involves simultaneously optimizing two or more, often conflicting, objective functions. Improving one objective may lead to the deterioration of another, making it impractical to obtain a single solution that is optimal for all objectives. Instead, MOP aims to find a set of solutions that represent optimal trade-offs among the objectives. In this study, optimization is approached as the minimization of objective values.

Formally, a MOP can be stated as (Ehrgott 2005):

$$\text{Minimize} \quad \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x}))$$

Subject to

$$\mathbf{x} \in \Omega$$

Here, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a vector of n decision variables residing in the n-dimensional decision space $\mathbb{R}^n$. The set $\Omega \subseteq \mathbb{R}^n$ represents the feasible region of the decision space, containing all permissible solutions. In the context of community detection, $\mathbf{x}$ typically represents a candidate partitioning of the network's nodes into communities. The function $\mathbf{f} : \Omega \to \mathbb{R}^m$ maps the decision space to the m-dimensional objective space, where each $f_i(\mathbf{x})$ is an objective function to be minimized. The image of the feasible set, $\mathbf{f}(\Omega) = \{\mathbf{f}(\mathbf{x}) \mid \mathbf{x} \in \Omega\}$, is the set of all possible objective vectors in the objective space $\mathbb{R}^m$. The problem can also be subject to additional constraints in Ω and/or $\mathbf{f}(\Omega)$.

Since the objective space $\mathbb{R}^m$ (for $m > 1$) is only partially ordered, the concept of dominance is used to compare solutions. Let $\mathbf{a} = (a_1, a_2, \ldots, a_m)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_m)$ be two objective vectors in $\mathbb{R}^m$. The vector $\mathbf{a}$ is said to dominate $\mathbf{b}$ (denoted as $\mathbf{a} \prec \mathbf{b}$) if and only if $a_i \leq b_i$ for all $i \in \{1, \ldots, m\}$ and $a_j < b_j$ for at least one $j \in \{1, \ldots, m\}$. A solution $\mathbf{x}^* \in \Omega$ is Pareto optimal if there is no other solution $\mathbf{x} \in \Omega$ such that $\mathbf{f}(\mathbf{x})$ dominates $\mathbf{f}(\mathbf{x}^*)$. In other words, $\mathbf{x}^*$ is Pareto optimal if its corresponding objective vector $\mathbf{f}(\mathbf{x}^*)$ is non-dominated.

The set of all non-dominated objective vectors is known as the Pareto Front (PF), and the set of all Pareto optimal decision vectors is the Pareto Optimal set (PO):

$$PF = \{\mathbf{y} \in \mathbf{f}(\Omega) \mid \nexists \mathbf{y}' \in \mathbf{f}(\Omega) \text{ such that } \mathbf{y}' \prec \mathbf{y}\} \tag{4}$$

$$PO = \{\mathbf{x} \in \Omega \mid \mathbf{f}(\mathbf{x}) \in PF\} \tag{5}$$

The PF represents the set of optimal trade-off solutions where no objective can be improved without degrading at least one other objective (Moreira and Paquete 2019).

Multi-objective methods offer several advantages, particularly for community detection. First, they provide a diverse set of solutions, allowing users to select a partition based on context-specific priorities or further analysis. Second, the simultaneous consideration of complementary objectives can mitigate structural biases inherent in single-objective metrics and help avoid premature convergence to suboptimal solutions. Third, characteristics such as the number of detected communities can emerge naturally from the optimization process rather than being a predefined parameter. Finally, the exploration of multiple trade-off solutions often reveals richer structural insights into the network, potentially highlighting overlapping or hierarchical community structures.

Given the typically non-convex and highly combinatorial nature of the community detection problem, obtaining the true PF and PO is computationally challenging. Deterministic optimization methods are often impractical. Consequently, Multi-Objective Evolutionary Algorithms (MOEAs) have become a prominent strategy (Deb 2001; Handl and Knowles 2007; Coello et al. 2007). MOEAs are well-suited for exploring complex search spaces due to their population-based approach and stochastic

operators. While they do not guarantee finding the exact PF and PO sets, MOEAs aim to produce a finite set of non-dominated solutions that effectively approximate the true Pareto front.[1]

Among the earliest applications of evolutionary multi-objective optimization in this domain is **MOGA-Net**, introduced by Pizzuti (2009). Her approach uses genetic algorithms to evolve network partitions that optimize both a community score and modularity. Although pioneering, the method remains dependent on modularity as a selection criterion, which reintroduces resolution bias. Additionally, the algorithm's computational cost is not thoroughly characterized, leaving its applicability to large networks uncertain.

Building on the general definition of MOP, the Multi-Objective Community Detection (MOCD) algorithm of Shi et al. (2010) can be formulated as follows. Let $C = \{c_1, c_2, \ldots, c_k\}$ be a partition of the node set V of a graph into k communities. The set of all valid partitions of V is denoted as the feasible set $\Omega = \{C \mid C \text{ is a valid partition of } V\}$. The MOCD formulation utilizes a vector objective function $\mathbf{f}(C)$ comprising two objectives:

$$f_1(C) = 1 - \sum_{c \in C} \frac{|E(c)|}{m}, \quad f_2(C) = \sum_{c \in C} \Big(\frac{\sum_{v \in c} \deg(v)}{2m} \Big)^2. \tag{6}$$

In these expressions, $m = |E|$ represents the total number of edges in the network, $\deg(v)$ denotes the degree of node v, and $|E(c)|$ is the number of edges with both endpoints lying within community c.

The MOCD problem is then stated as:

$$\min_{C \in \Omega} \mathbf{f}(C) = \min_{C \in \Omega} \big(f_1(C),\, f_2(C)\big).$$

To approximate the Pareto Optimal set, MOCD employs the Pareto Envelope-based Selection Algorithm II (PESA-II), as proposed by Corne et al. (2000). This evolutionary algorithm maintains both an internal population for exploration and an external archive of non-dominated solutions. Its encoding scheme is based on locus-based adjacency, which ensures that communities remain connected; however, this restricts the search space to particular types of partitions. While the use of local mutation and standard two-point crossover operators is computationally simple, it can potentially reduce genetic diversity and disrupt meaningful substructures within the solutions.

In terms of computational cost, the algorithm evaluates fitness in $O(m + n)$ time per solution, where $n = |V|$ is the number of nodes. PESA-II introduces an additional computational overhead of $O(gs^2(m + n))$ across g generations with a population size of s. Furthermore, MOCD applies a Max–Min Distance selection strategy to promote diversity among solutions. This step, however, increases runtime due to the necessity of pairwise comparisons against randomly generated control solutions. Consequently, MOCD may not scale efficiently to large-scale networks.

[1] The proximity of the set of solutions obtained by an MOEA to the true PF is a measure of the algorithm's convergence.

Overall, while existing multi-objective algorithms demonstrate the theoretical advantages of Pareto optimization, they remain under-explored in terms of implementation quality, scalability, and general usability. This highlights the need for more efficient and accessible methods, particularly in large-scale applications, a gap that this work aims to address.

3 The HP-MOCD Algorithm

This section details the High-Performance Multi-Objective Community Detection (HP-MOCD) algorithm, a scalable evolutionary method for efficiently identifying high-quality community partitions in large complex networks. HP-MOCD combines the NSGA-II optimization framework with a parallel architecture and topology-aware genetic operators tailored to the structure of real-world graphs. In addition to detailing its core components, we describe the algorithm's design choices, solution representation, and multi-objective selection strategy. The implementation is written in Rust for performance and exposed to Python via PyO3. The full source code is publicly available on GitHub.

3.1 Overview and Design Rationale

Let us consider a network represented by $G = (V, E)$. The goal of HP-MOCD is to uncover community structures in G by minimizing the two objectives proposed by (Shi et al. 2010) in Equation 6. By jointly optimizing these criteria, the algorithm promotes partitions in which communities are internally cohesive and externally well-separated, balancing internal density with structural isolation. To accomplish this, the algorithm builds upon the NSGA-II framework, a widely adopted and effective method in the field of multi-objective evolutionary algorithms. NSGA-II was chosen primarily due to its strong performance in producing well-distributed and convergent Pareto fronts, especially when compared to earlier algorithms like PESA-II (Diosan and Oltean 2007), which tend to suffer from loss of diversity or weaker selection pressure.

The HP-MOCD procedure unfolds in two main phases. First, it constructs an initial population of potential solutions (called individuals), each representing a possible way to partition the graph into communities. Then, it evolves this population over a number of generations using genetic operators, continuously selecting and refining the best trade-offs between the objectives. This process is summarized in Algorithm 1.

Internally, the graph G is stored using a high-performance hash map that maps each node to its list of neighbors. This representation, based on a fast, non-cryptographic hashing scheme [2], enables efficient access and mutation operations during the evolution cycle, which is especially important for large-scale networks. In the subsequent phase, the NSGA-II algorithm produces a set of non-dominated solutions that collectively form a Pareto front. A Pareto front comprises multiple optimal solutions, each offering a different trade-off between the two conflicting objectives. A solution, commonly referred to as an *individual* in evolutionary algorithms, is specifically represented as a mapping (hashmap) that associates each node of the graph with a particular community

[2] A custom hash algorithm was used by rustc (plus hashmap/set aliases): fast, deterministic, not secure https://github.com/rust-lang/rustc-hash

Algorithm 1 Overview of the HP-MOCD Algorithm.

procedure HP-MOCD(Graph G, Population Size N, Max Generations T, Crossover Probability C_P, Mutation Probability M_P)
    $P \leftarrow$ InitializePopulation(G, N)
    EvaluatePopulation(P, G) ▷ (Equation 6)
    for $gen \leftarrow 0$ **to** T **do**
        ComputeCrowdingDistance(P)
        $M \leftarrow$ TournamentSelection(P, N) ▷ By rank & crowding
        $Q \leftarrow$ CreateOffSpring$(M,\ G,\ C_P,\ M_P, 4)$ ▷ (Algorithm 2)
        EvaluatePopulation$(Q,\ G)$
        $R \leftarrow P \cup Q$
        $P \leftarrow$ SelectNextGeneration(R, N) ▷ Keep best N by rank + crowding
    end for
    $\mathcal{F}_1 \leftarrow \{\, x \in P \mid \text{rank}(x) = 1\}$
    return $\mathcal{F}_1$
end procedure

identifier. Thus, each individual represents a unique community partitioning of the graph.

To better illustrate this concept, consider the toy network shown in Figure 1. Based on this graph, HP-MOCD may generate many different community configurations over time. These solutions reflect different compromises between internal density and external sparsity, and none is necessarily superior to the others in all aspects.

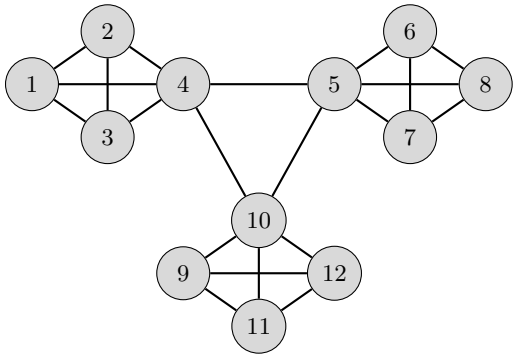


Fig. 1 A simple network with some nodes that shows aspects of a community.

All these individuals belong to the final Pareto front produced by the algorithm. Since no single partition can simultaneously minimize both objectives without compromising the other, the algorithm instead returns a diverse set of non-dominated solutions. Each one represents a valid alternative, offering different structural insights. An individual (solution) produced by our algorithm is stored as a mapping,

$$\mathcal{M} : V \longrightarrow \{A, B, C, \dots\},$$

For example, consider the decoded individual for the graph in Figure 1:

$$\mathcal{M} = \{\,1{:}A,\ 2{:}A,\ 3{:}A,\ 4{:}A,\ 5{:}B,\ 6{:}B,\ 7{:}B,\ 8{:}B,\ 9{:}C,\ 10{:}C,\ 11{:}C,\ 12{:}C\}.$$

If we choose $\phi(A) =$ blue, $\phi(B) =$ green, and $\phi(C) =$ red, then nodes $\{1, 2, 3, 4\}$ appear in blue, $\{5, 6, 7, 8\}$ in green, and $\{9, 10, 11, 12\}$ in red, as shown in Figure 2. And this can be just one of a large set of solutions that the proposed HP-MOCD can return to us.

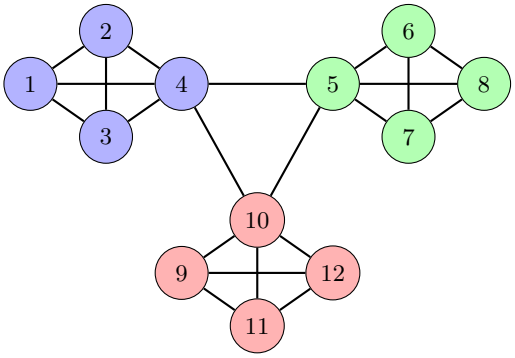


Fig. 2 The same network, but with distinct communities. Nodes within each community (blue, green, and red) are more densely connected internally, with sparser connections between communities.

This example illustrates a key advantage of the multi-objective approach: instead of converging to a single solution, HP-MOCD yields a diverse set of high-quality alternatives. This flexibility allows to select the partition that best fits the specific needs or constraints of their application domain.

3.2 Genetic Operators and Offspring Generation

3.2.1 Crossover

The crossover operator is responsible for combining multiple parent solutions to generate a new offspring. Let $P = (P_1, \ldots, P_n)$ denote a collection of n parent individuals, where each individual P_i represents a partition of the graph $G = (V, E)$, that is, a mapping from each node $v \in V$ to a community label $c \in C$. The operator is controlled by a crossover threshold parameter $C_R \in [0, 1]$. Initially, a random number x is sampled uniformly from the interval $[0, 1]$. If $x > C_R$, the crossover is skipped, and one of the parent partitions is selected uniformly at random to serve as the offspring:

$$x > C_R \quad \Rightarrow \quad P_{\text{child}} = \text{Random}(P_1, \ldots, P_n). \tag{7}$$

If $x \leq C_R$, the offspring is constructed by aggregating community assignments from all parent individuals. The process starts by considering the common node set V, and initializing an empty offspring mapping P_{child}. For each node $v \in V$, the algorithm

counts the number of times each community label $c \in C$ has been assigned to v across all parent individuals, using the following expression:

$$\text{count}(c, v) = \sum_{i=1}^{n} \mathbf{1}\{P_i(v) = c\}, \tag{8}$$

where $\mathbf{1}\{\cdot\}$ denotes the indicator function. The community (or communities) with the highest count is then identified as

$$c^*(v) \in \arg\max_{c \in C} \text{count}(c, v). \tag{9}$$

In the event of a tie, one of the top-scoring communities is selected uniformly at random. The selected label $c^*(v)$ is then assigned to node v in the offspring partition, that is, $P_{\text{child}}(v) = c^*(v)$. This procedure is applied to all nodes in V, producing a complete offspring partition. The resulting solution tends to preserve dominant structural features from the parents while still allowing for variability and exploration through random tie-breaking.

3.2.2 Mutation

To introduce variation and avoid premature convergence, HP-MOCD applies a mutation operator after crossover. This operator modifies the community assignment of selected nodes based on the structure of their local neighborhood. The mutation operator uses the same individual representation as described in Section 3.2.1, along with a mutation threshold parameter $M_R \in [0, 1]$. Each node $v \in V$ is considered for mutation independently: a random number $r_v \in [0, 1]$ is sampled, and nodes satisfying $r_v < M_R$ are selected for mutation. The set of selected nodes is defined as:

$$V_m = \{v \in V \mid r_v < M_R\}. \tag{10}$$

For each selected node, the frequency of community assignments among its neighboring nodes is computed. To improve performance, the nodes are processed in batches. For a given node v, its set of neighbors $N(v)$ is retrieved, and for each neighbor $u \in N(v)$, the community to which u belongs is noted. The algorithm then determines the most frequent community using the same Equation 9.

The community assignment of v is updated to this most common community. After all selected nodes have been processed, the original mapping is updated with these new community assignments. The Offspring Generation procedure, which is presented in detail in Algorithm 2, is performed as follows.

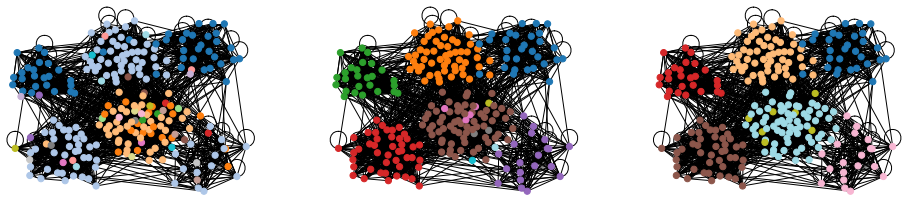
3.2.3 Parameters

The evolutionary process in HP-MOCD is governed by two key hyperparameters: population size and number of generations. To determine appropriate values, we conducted empirical experiments on synthetic benchmarks, using the same configuration as in our main experiments, except for varying n (see Section 4.1.1). We analyzed the

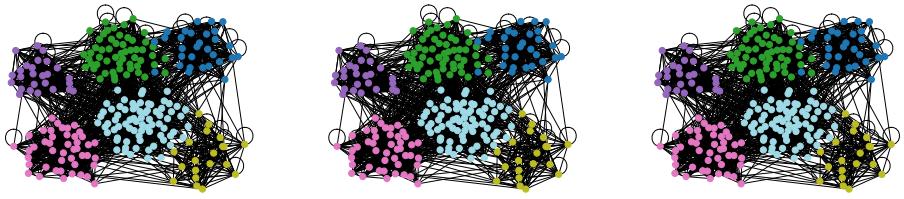
Algorithm 2 Offspring Generation

procedure CREATEOFFSPRING(M, G, C_P, M_P, E_S)
  $Q \leftarrow [\,]$ ▷ Initialize offspring population
  for $i \leftarrow 1$ **to** $|M|$ **do**
    $x \leftarrow \text{random}(0, 1)$
    if $x < C_P$ **then** ▷ Use E_s parents to crossover
      $Parents \leftarrow \text{SelectRandom}(M, E_S)$
      $P_{child} \leftarrow \text{Crossover}(Parents)$
    else
      $P_{child} \leftarrow \text{Random}(M)$
    end if
    $\text{Mutate}(P_{child}, G, M_P)$
    Append $P_{child} \rightarrow Q$
  end for
  return Q
end procedure

convergence behavior of the algorithm under these settings. As shown in Figure 3, 100 generations provided a favorable trade-off between solution quality and computational cost; for the other parameters and scenarios, we will leave them in the supplementary material (See 5).



(a) 10 Generations (b) 30 Generations (c) 50 Generations



(d) 80 Generations (e) 100 Generations (f) 110 Generations

Fig. 3 Performance results across varying *maxgen* values with $\mu = 0.1$. Refer to Section 4.1.1 for details on the experimental setup.

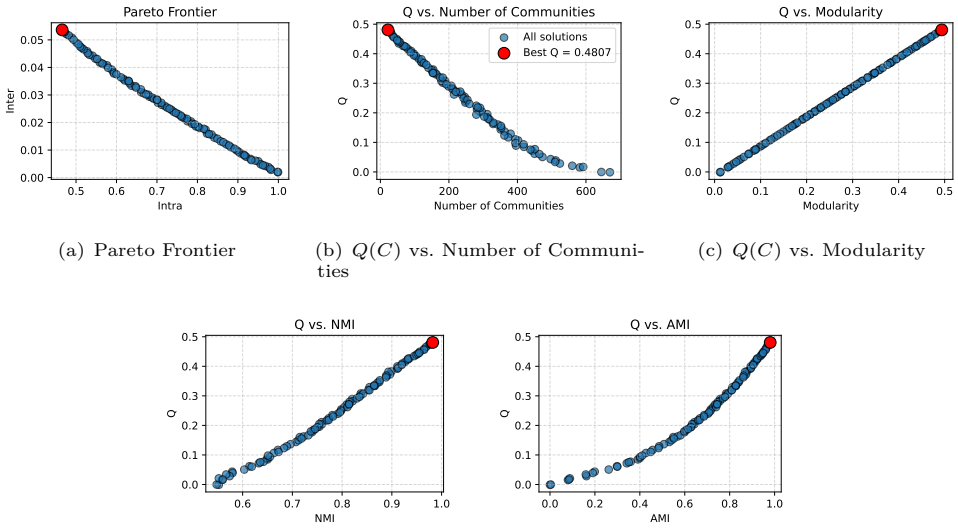
3.3 Pareto Selection

As discussed in Section 2, multi-objective optimization algorithms such as NSGA-II do not return a single "optimal" solution. Instead, they produce a *Pareto front*, a set of non-dominated solutions, in which no solution is strictly better than another across all objectives. This naturally raises the question: how can one select the most appropriate solution from the Pareto front?

Following the strategy proposed by Shi et al. (2010), we adopt a scalarization-based criterion to identify the solution that maximizes the following quality function:

$$Q(C) = 1 - \text{intra}(C) - \text{inter}(C) \tag{11}$$

This score favors solutions with low intra-community penalty and low inter-community connectivity, as defined in Equations 6, thus promoting partitions that are both cohesive and well-separated. Figure 4 illustrates this behavior: the red dot indicates the solution that maximizes $Q(C)$. We observe that this selection strategy consistently identifies partitions that also perform well under widely used external validation metrics such as modularity, NMI, and AMI.



(a) Pareto Frontier

(b) $Q(C)$ vs. Number of Communities

(c) $Q(C)$ vs. Modularity

(d) $Q(C)$ vs. NMI

(e) $Q(C)$ vs. AMI

Fig. 4 Analysis of the scalarization-based selection method. The red dot highlights the solution with the highest value of $Q(C)$.

An important advantage of the Pareto-based approach is the flexibility it affords in selecting solutions based on application-specific preferences. While we adopt the scalarization function $Q(C)$ as a default selection criterion, practitioners are not constrained to this choice. Depending on the context, alternative metrics can be employed to guide selection from the Pareto front. For instance, one might prioritize

solutions based on modularity, the number of detected communities, average community size, or even the variance in community sizes. Other structural properties, such as conductance, cut ratios, or expansion scores, may also be relevant in certain domains. This flexibility allows users of HP-MOCD to serve as a general-purpose tool, as it provides not just one partition, but a rich set of candidate solutions from which the most appropriate can be chosen using auxiliary metrics or domain-specific constraints. This is a fundamental departure from single-objective methods, which collapse the search space into a single score and, therefore, offer limited interpretability and adaptability.

3.4 Complexity Analysis

Complexity time of HP-MOCD per generation is driven by its genetic operators (crossover and mutation) and the key NSGA-II routines (non-dominated sorting and crowding distance). Let $|V|$ be the number of nodes, $|E|$ the number of edges, N_p the population size, and M_R the mutation rate.

3.4.1 Operators Complexity Analysis

Crossover & Mutation

For each node in the graph, the algorithm crossover identifies the most frequent community label assigned to that node among a fixed number of parent solutions. Since hash map operations have an expected constant time complexity, $O(1)$, processing each node takes $O(1)$ time. Assigning the most frequent label also takes constant time per node. Consequently, for all $|V|$ nodes, the expected time complexity of a single crossover operation is linear in the number of nodes, i.e., $O(|V|)$.

The mutation iterates through each of the $|V|$ nodes, selecting a node for mutation with a given probability $M_R \in [0, 1]$. This selection phase involves a random draw for each node, costing $O(|V|)$ time, and, for each node selected for mutation, inspects its local neighborhood to determine a new community assignment (by finding the most frequent community label among its neighbors). This requires visiting all adjacent nodes, so the cost for mutating a single node v is proportional to its degree, $d(v)$. The expected number of nodes selected for mutation is $M_R \cdot |V|$. The total time for recalculating assignments for these mutated nodes is proportional to the sum of their degrees. In expectation, this sum is $M_R \sum_{v \in V} d(v) = M_R \cdot 2|E|$. Therefore, the expected time complexity for this part of the mutation is $O(M_R \cdot |E|)$. Combining the node selection phase and the assignment update phase, the total time complexity for one mutation operation is $O(|V| + M_R \cdot |E|)$. For sparse graphs, where $|E| = O(|V|)$, and assuming M_R is a constant, this simplifies to $O(|V|)$.

Non-Dominated Sorting & Crowding Distance Calculation

These operators act upon a combined population of $2N_p$ individuals (current population P_t and offspring population Q_t, each of size N_p). This process sorts the $2N_p$ individuals into different non-domination fronts. The non-dominated sorting of N individuals can be performed in $O(N \log N)$ time with fixed 2 objectives. Thus, for the $2N_p$ individuals, this step takes $O((2N_p) \log(2N_p)) = O(N_p \log N_p)$ time.

For each front of size F_i, solutions are sorted based on each of the M objective functions. This takes $O(MF_i \log F_i)$ time per front. Summing over all fronts $\sum F_i = 2N_p$, the total time complexity for crowding distance calculation across all fronts is $O(M(2N_p)\log(2N_p))$. For $M = 2$, this becomes $O(N_p \log N_p)$.

Overall Complexity

The dominant cost per generation is the sum of: $T_{\text{gen}} = O(N_p|V|) + O(N_p \log N_p) + O(N_p \log N_p)$ which simplifies to $T_{\text{gen}} = O(N_p|V| + N_p \log N_p)$. Since $|V|$ in G is typically much larger than $\log N_p$, the $O(N_p|V|)$ term usually dominates. Thus, the complexity per generation effectively becomes: $O(N_p|V|)$ for sparse graphs. Over G generations, the total time complexity of HP-MOCD is: $O(G \cdot N_p|V|)$.

4 Experiments

This section presents a series of experiments designed to assess the performance of the proposed HP-MOCD algorithm in large-scale networks. Our goal is to evaluate both the quality of the detected communities and the scalability of the method in terms of runtime. All experiments are fully reproducible, with code and data generation scripts publicly available on GitHub.

4.1 Evaluation Setup

To evaluate the performance of the proposed HP-MOCD algorithm, we conducted a set of experiments using synthetic benchmark networks. This setup enables the controlled variation of structural properties, such as size, noise, and community separability, while providing known ground-truth partitions for external validation.

4.1.1 Synthetic Benchmark Networks

We used the LFR benchmark generator (Lancichinetti et al. 2008), implemented via the NetworkX Python library, to generate realistic artificial networks with built-in community structure. These benchmarks are widely used due to their ability to replicate real-world network features, such as power-law degree distributions and heterogeneous community sizes. Two main experimental settings were considered:

- **Scalability test:** The number of nodes n varied from 10,000 to 100,000, while all other parameters remained fixed. This setup evaluates how each algorithm scales computationally and structurally with increasing network size.
- **Robustness test:** With network size fixed at $n = 100{,}000$, the mixing parameter $\mu \in [0.1, 0.8]$ was varied. This parameter controls the proportion of each node's links that connect to other communities. Smaller values of μ correspond to well-separated communities, while higher values represent noisy, overlapping structures that are harder to detect.

All LFR graphs were generated using the following fixed parameters: maximum degree = 50, average degree = 20, minimum community size = 20, maximum community

size = 100, degree exponent $\tau_1 = 2.5$, and community size exponent $\tau_2 = 1.5$. It is important to note that in all generated synthetic networks, the proportion of overlapping nodes and the number of communities per overlapping node were both set to zero. This choice reflects our focus on non-overlapping community detection. The known ground truth structure of these LFR benchmark networks allowed us to directly measure and evaluate the accuracy and effectiveness of the HP-MOCD algorithm.

4.1.2 Algorithms and Parameter Configuration

We compared HP-MOCD to two multi-objective evolutionary algorithms (MOEAs) and two classical greedy heuristics:

- **MOCD** (Shi et al. 2010): Optimizes intra-community density and inter-community sparsity using the PESA-II framework.
- **MOGA-Net** (Pizzuti 2009): A modularity-based evolutionary algorithm with population-based search.
- **Louvain** (Blondel et al. 2008): Greedy modularity maximization, widely used for its speed and simplicity.
- **Leiden** (Traag et al. 2019): Improves Louvain by ensuring connected communities and more stable refinement.

All algorithms were configured based on their original publications. HP-MOCD used the following configuration, tuned empirically based on preliminary trials (see Section 3.2.3): crossover probability $C_P = 0.8$, mutation probability $M_P = 0.2$, and number of parents per crossover $E_S = 4$. Table 1 summarizes the configurations.

Table 1 Parameter settings and types of the compared algorithms. C_P and M_P denote crossover and mutation probabilities, and E_S is the number of parents per crossover.

Algorithm	Type	Generations	Population	C_P	M_P	E_S	Reference
MOCD	MOEA	100	100	0.9	0.1	-	(Shi et al. 2010)
MOGA-Net	MOEA	100	100	0.9	0.1	-	(Pizzuti 2009)
Louvain	Greedy	-	-	-	-	-	(Blondel et al. 2008)
Leiden	Greedy	-	-	-	-	-	(Traag et al. 2019)
HP-MOCD	MOEA	100	100	0.8	0.2	4	Ours

All algorithms were implemented in Python, with the exception of HP-MOCD, which is implemented in Rust and exposed to Python via PyO3[3]. Although HP-MOCD benefits from parallel execution, none of the evaluated algorithms exploit parallelism in their algorithmic logic (e.g., population-level operators), making runtime comparisons consistent.

[3] https://pyo3.rs/v0.25.0/

4.1.3 Evaluation Metrics

We evaluate performance using three complementary metrics:

Normalized Mutual Information (NMI):

This metric quantifies the agreement between the detected community partition C and the ground-truth partition C' (Yao 2003; Vinh et al. 2010), defined as:

$$\mathrm{NMI}(C, C') = \frac{-2 \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} M_{ij} \log\left(\frac{M_{ij} N}{M_{i\cdot} M_{\cdot j}}\right)}{\sum_{i=1}^{n_1} M_{i\cdot} \log\left(\frac{M_{i\cdot}}{N}\right) + \sum_{j=1}^{n_2} M_{\cdot j} \log\left(\frac{M_{\cdot j}}{N}\right)}, \tag{12}$$

where M_{ij} is the confusion matrix between partitions, N is the number of nodes, and $M_{i\cdot}$, $M_{\cdot j}$ are marginal sums. NMI ranges from 0 (no agreement) to 1 (perfect match).

Adjusted Mutual Information (AMI):

AMI corrects the mutual information for chance (Vinh et al. 2010), offering greater robustness when the number or size of communities varies:

$$\mathrm{AMI}(U, V) = \frac{\mathrm{MI}(U, V) - \mathbb{E}[\mathrm{MI}(U, V)]}{\max(H(U), H(V)) - \mathbb{E}[\mathrm{MI}(U, V)]}, \tag{13}$$

where $\mathrm{MI}(U, V)$ is the mutual information between partitions U and V, $\mathbb{E}[\mathrm{MI}]$ is its expected value under random labeling, and $H(\cdot)$ denotes entropy.

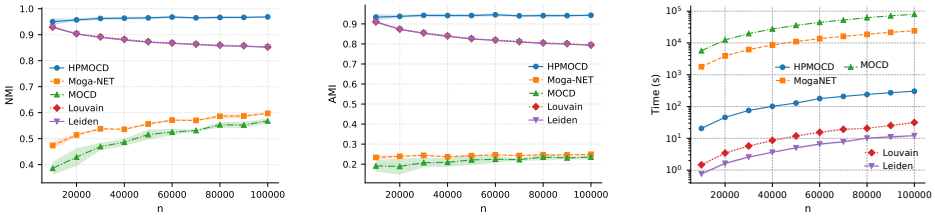
4.1.4 Execution Environment

All experiments were executed on a Linux server equipped with a single Intel Core i7-4790 CPU @ 3.60 GHz (4 physical cores, 2 threads per core, 8 threads total) and 16 GB of RAM. HP-MOCD employed multithreaded execution, while all other algorithms were run in single-threaded mode to ensure a fair comparison. Each reported result corresponds to the mean over 20 independent runs initialized with different random seeds.

4.2 Impact of Network Size on Performance

Figure 5 presents a comparative evaluation of the algorithms' performance across synthetic networks of increasing size, ranging from 10,000 to 100,000 nodes, with a fixed mixing parameter of $\mu = 0.3$. Each result corresponds to the average over 20 independent runs, each initialized with a different random seed to account for stochastic variability and improve robustness. For all experiments, 95% confidence intervals were also computed to capture variability between runs.

The results show a clear advantage for HP-MOCD in terms of both detection quality and computational scalability. In Figures 5(a) and (b), HP-MOCD consistently achieves high NMI and AMI scores across all tested network sizes. Its detection quality remains stable and close to optimal, with NMI values near 1.0 and AMI values exceeding 0.9, indicating a strong alignment with the ground truth community structure. This



(a) NMI Value (b) AMI Value (c) Execution Time (s)

Fig. 5 AMI, NMI, and runtime (seconds) comparison of algorithms on LFR benchmark networks of different node counts (n) with a fixed mixing parameter ($\mu = 0.3$), averaged over 20 independent runs.

consistency highlights HP-MOCD's robustness to increasing graph size and suggests that the algorithm does not degrade as the problem scales. In contrast, the other MOEA-based methods (MOGA-Net and MOCD) demonstrate considerably lower accuracy at larger scales. Both exhibit modest improvements in NMI as n increases, but plateau at significantly lower levels, and their AMI scores remain markedly below those of HP-MOCD. This confirms that the optimization strategies and genetic operators employed in HP-MOCD offer a more effective exploration of the solution space.

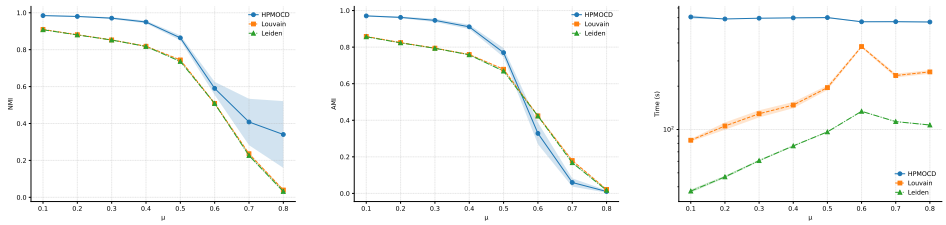
Figure 5(c) reveals the effects of network size on runtime. Note that the y-axis is presented on a logarithmic scale to accommodate the wide disparity in execution times. While the greedy algorithms (Louvain and Leiden) maintain a sublinear scaling as expected, their detection quality remains consistently inferior to that of HP-MOCD. Among the MOEA-based approaches, HP-MOCD exhibits a much lower runtime—up to many orders of magnitude faster than MOCD and significantly faster than MOGA-Net. This improvement can be directly attributed to HP-MOCD's parallel architecture and efficient implementation. Importantly, when the network size exceeds 40,000 nodes, MOCD becomes impractical due to its steep runtime increase, whereas HP-MOCD remains computationally viable. These findings emphasize the algorithm's scalability and practical utility in real-world settings where large graphs are common. In summary, HP-MOCD achieves the best balance between scalability and detection accuracy among the evaluated methods.

4.3 Resilience to Community Mixing

Figure 6 presents the performance of the algorithms under increasing levels of structural noise, as controlled by the mixing parameter μ in the LFR benchmark networks. Only algorithms with satisfactory performance from the previous experiment were included in this phase. MOCD and MogaNET algorithms that performed poorly at $\mu = 0.3$ were excluded, as their performance is expected to deteriorate further with higher noise and their prohibitively long execution times, which exceeded 25 hours for a single run, making them impractical for this analysis.

Higher values of μ correspond to a greater proportion of edges connecting nodes across different communities, thereby reducing the separability of community structure and making the detection task more difficult. All experiments were conducted on

networks with 100,000 nodes, averaged over 20 independent runs with different seeds. Results include 95% confidence intervals.



(a) NMI Value (b) AMI Value (c) Execution Time (s)

Fig. 6 AMI, NMI and runtime (s) of the algorithms under comparison on LFR networks with different values of the mixing parameter μ and 100,000 nodes, averaged over 20 independent runs.

As shown in Figures 6(a) and (b), the proposed HP-MOCD algorithm achieves superior detection quality across most values of μ. Both NMI and AMI remain high and stable up to $\mu = 0.5$, indicating strong resilience to moderate levels of noise and community overlap. This suggests that HP-MOCD maintains accurate partitions even when the boundary between communities becomes less distinct. As expected, all algorithms decline in performance for $\mu \geq 0.6$, where communities are heavily intermixed. In this regime, the structural signal weakens, and the task of community detection approaches the limit of observability. Though HP-MOCD also degrades beyond this point, it consistently outperforms Louvain and Leiden in NMI and AMI, especially for $\mu = 0.6$ and $\mu = 0.7$, where the difference remains substantial.

The runtime analysis in Figure 6(c) shows that HP-MOCD maintains stable execution times across all tested values of μ, demonstrating robustness to changes in edge density. In contrast, Louvain and Leiden exhibit more variability, including notable runtime spikes near $\mu = 0.6$, likely due to instability in the community structure causing convergence delays. Despite being a multi-objective evolutionary algorithm, HP-MOCD offers competitive and consistent runtime performance. This is due to the linear expected time complexity of its genetic operators regarding the number of edges, allowing the algorithm to scale efficiently even as network complexity increases as detailed in Section 3.

Overall, these results confirm that HP-MOCD is well-suited for detecting communities in networks with varying degrees of structural clarity. Its resilience to increasing μ highlights its ability to balance multiple structural criteria without succumbing to premature convergence or overfitting to noise.

4.4 Discussion

Experimental results highlight HP-MOCD's performance, scalability, and robustness. First, with increasing network sizes (Figure 5), HP-MOCD maintained high detection quality and showed a clear computational advantage over other evolutionary algorithms. It was the only EA-based method to process networks over 40,000 nodes within a

reasonable time. The greedy heuristics (Louvain, Leiden) were faster, but mostly produced lower quality partitions, which was confirmed by the NMI and AMI values. This highlights HP-MOCD's strong parallelized architecture, enabling effective scaling while preserving detection accuracy.

Second, in resilience tests with the mixing parameter μ (Figure 6), HP-MOCD outperformed baselines in moderate-noise conditions, up to approximately $\mu = 0.6$. In this regime, its detection quality remained high, indicating its ability to uncover community structures even in networks with significant inter-community connectivity. Practically, HP-MOCD offers several unique advantages. Unlike traditional methods that return a single solution, it produces a diverse Pareto front of high-quality alternatives. This allows practitioners to select community partitions based on application-specific criteria, such as the desired number of communities, modularity, or community size distribution. Its flexibility in balancing multiple structural objectives is valuable for real-world, context-dependent trade-offs.

Despite these strengths, several limitations remain. While robust across increasing n and μ, further studies should assess its behavior in networks with other structural properties (e.g., varying degree distributions, hierarchical structures, overlapping communities). Lastly, while AMI and NMI assessed partition quality, they still have shortcomings. NMI can inflate scores, while AMI provides reliable corrections for chance. However, the main conclusions drawn from these experiments are consistent across both metrics, reinforcing the validity of the observed trends. Overall, HP-MOCD shows promise as a high-quality, scalable solution for multi-objective community detection, especially in large and moderately noisy networks, ideal for social networks.

5 Conclusion and Future Work

We introduced HP-MOCD, a high-performance evolutionary multi-objective algorithm for large-scale network community detection. Built on the NSGA-II framework, HP-MOCD incorporates topology-aware crossover and mutation operators, coupled with a parallelized, efficient implementation, ensuring scalability and practical applicability. Unlike traditional single-objective heuristics and earlier MOEA-based approaches, HP-MOCD provides a principled mechanism to capture diverse structural trade-offs via Pareto-optimal partitions. The algorithm consistently outperformed other evolutionary methods in runtime and scalability, maintaining competitive or superior accuracy, even in complex or noisy networks.

These results significantly advance the state-of-the-art in multi-objective community detection. While most existing methods are limited by high computational costs or inflexibility, HP-MOCD demonstrates multi-objective expressiveness combined with high-performance execution. Furthermore, its ability to yield diverse, high-quality partitions makes the algorithm especially suitable for exploratory network analysis and downstream tasks benefiting from structural alternatives.

Future work can extend the algorithm to support overlapping communities and more general network types, such as weighted, directed, or temporal graphs. Another promising path involves improving robustness under high mixing regimes ($\mu \to 1$), potentially through hybrid strategies or advanced selection mechanisms. Finally, we

intend to explore integration with NSGA-III (Deb and Jain 2014), and other recent MOEA frameworks to further improve performance in scenarios involving more than two objectives, such as modularity, conductance, or community balance.

Acknowledgements

This work was supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq, grant 307151/2022-0, 445324/2024-4), Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG, grant APQ-01647-22). We also thank the Universidade Federal de Ouro Preto (UFOP) for its support.

References

Azevedo, B.F., Rocha, A.M.A.C., Fernandes, F.P., Pacheco, M.F., Pereira, A.I.: Comparison between single and multi-objective clustering algorithms: Mathe case study. In: Optimization, Learning Algorithms and Applications, pp. 65–80. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-77426-3_5

Blondel, V.D., Guillaume, J.-L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment **2008**(10), 10008 (2008) https://doi.org/10.1088/1742-5468/2008/10/P10008

Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., Hwang, D.-U.: Complex networks: Structure and dynamics. Physics Reports **424**(4), 175–308 (2006) https://doi.org/10.1016/j.physrep.2005.10.009

Corne, D.W., Knowles, J.D., Oates, M.J.: The pareto envelope-based selection algorithm for multiobjective optimization. In: Parallel Problem Solving from Nature PPSN VI (2000). https://doi.org/10.1007/3-540-45356-3_82

Coello, C.A.C., Lamont, G.B., Veldhuizen, D.A.V.: Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation). Springer, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-0-387-36797-2

Deb, K.: Multiobjective Optimization Using Evolutionary Algorithms. John Wiley & Sons, New York (2001). https://www.wiley.com/en-us/Multi+Objective+Optimization+using+Evolutionary+Algorithms-p-9780471873396

Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. IEEE Transactions on Evolutionary Computation **18**(4), 577–601 (2014) https://doi.org/10.1109/TEVC.2013.2281535

Diosan, L., Oltean, M.: Who's better? pesa or nsga ii? In: Seventh International Conference on Intelligent Systems Design and Applications (ISDA 2007), pp. 869–874 (2007). https://doi.org/10.1109/ISDA.2007.106

Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002) https://doi.org/10.1109/4235.996017

Ehrgott, M.: Multicriteria Optimization, 2nd edn. Lecture Notes in Economics and Mathematical Systems, vol. 491. Springer, Berlin, Heidelberg (2005). https://doi.org/10.1007/3-540-27659-9 . Second edition

Fortunato, S., Barthélemy, M.: Resolution limit in community detection. Proceedings of the National Academy of Sciences **104**(1), 36–41 (2007) https://doi.org/10.1073/pnas.0605965104

Fortunato, S., Hric, D.: Community detection in networks: A user guide. Physics Reports **659**, 1–44 (2016) https://doi.org/10.1016/j.physrep.2016.09.002

Flake, G.W., Lawrence, S., Giles, C.L., Coetzee, F.M.: Self-organization and identification of web communities. Computer (2002) https://doi.org/10.1109/2.989932

Fortunato, S.: Community detection in graphs. Physics Reports **486**(3), 75–174 (2010) https://doi.org/10.1016/j.physrep.2009.11.002

Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proceedings of the National Academy of Sciences **99**(12), 7821–7826 (2002) https://doi.org/10.1073/pnas.122653799

Hussein, E., Hafez, A., Hassanien, A.E., Fahmy, A.: A discrete bat algorithm for the community detection problem, vol. 9121 (2015). https://doi.org/10.1007/978-3-319-19644-2_16

Handl, J., Knowles, J.: An evolutionary approach to multiobjective clustering. Evolutionary Computation, IEEE Transactions on **11**, 56–76 (2007) https://doi.org/10.1109/TEVC.2006.877146

Lancichinetti, A., Fortunato, S.: Community detection algorithms: A comparative analysis. Phys. Rev. E **80** (2009) https://doi.org/10.1103/PhysRevE.80.056117

Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Phys. Rev. E **78**, 046110 (2008) https://doi.org/10.1103/PhysRevE.78.046110

Moreira, G., Paquete, L.: Guiding under uniformity measure in the decision space. In: 2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI), pp. 1–6 (2019). https://doi.org/10.1109/LA-CCI47412.2019.9037034

Newman, M.E.J.: From the cover: Modularity and community structure in networks. Proceedings of the National Academy of Science **103**(23), 8577–8582 (2006) https://doi.org/10.1073/pnas.0601602103

Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E **69**, 026113 (2004) https://doi.org/10.1103/PhysRevE.69.026113

Pizzuti, C.: A multi-objective genetic algorithm for community detection in networks. In: 2009 21st IEEE International Conference on Tools with Artificial Intelligence, pp. 379–386 (2009). https://doi.org/10.1109/ICTAI.2009.58

Pizzuti, C.: A multiobjective genetic algorithm to find communities in complex networks. Evolutionary Computation, IEEE Transactions on **16**, 418–430 (2012) https://doi.org/10.1109/TEVC.2011.2161090

Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Phys. Rev. E **76**, 036106 (2007) https://doi.org/10.1103/PhysRevE.76.036106

Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., Parisi, D.: Defining and identifying communities in networks. Proceedings of the National Academy of Sciences **101**(9), 2658–2663 (2004) https://doi.org/10.1073/pnas.0400054101

Rossetti, G., Milli, L., Cazabet, R.: CDLIB: a Python Library to Extract, Compare and Evaluate Communities from Complex Networks. Applied Network Science **4**(1), 52 (2019) https://doi.org/10.1007/s41109-019-0165-9

Ravasz, E., Somera, A.L., Mongru, D.A., Oltvai, Z.N., Barabási, A.-L.: Hierarchical organization of modularity in metabolic networks. Science **297**(5586), 1551–1555 (2002) https://doi.org/10.1126/science.1073374

Shi, C., Zhong, C., Yan, Z., Cai, Y., Wu, B.: A multi-objective approach for community detection in complex network. In: IEEE Congress on Evolutionary Computation, pp. 1–8 (2010). https://doi.org/10.1109/CEC.2010.5585987

Traag, V.A., Waltman, L., Eck, N.J.: From Louvain to Leiden: guaranteeing well-connected communities. Scientific Reports **9**(1), 5233 (2019) https://doi.org/10.1038/s41598-019-41695-z

Vinh, N.X., Epps, J., Bailey, J.: Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. Journal of Machine Learning Research **11**, 2837–2854 (2010). http://jmlr.org/papers/v11/vinh10a.html

Wang, L., Zeng, Y., Li, Y., Liu, Z., Ma, J., Zhu, X.: Research on resolution limit of community detection in location-based social networks. In: 2019 International Conference on Networking and Network Applications (NaNA), pp. 90–95 (2019). https://doi.org/10.1109/NaNA.2019.00025

Yao, Y.Y.: Information-Theoretic Measures for Knowledge Discovery and Data Mining, pp. 115–136. Springer, Berlin, Heidelberg (2003). https://doi.org/10.1007/978-3-540-36212-8_6

Appendix A Space Complexity Analysis

Let $|V|$ represent the number of nodes, $|E|$ the number of edges, N_p the population size, and $d_{\max} = \max_{v \in V} d(v)$ the maximum node degree. The graph G itself is stored once, requiring $O\big(|V| + |E|\big)$ space. Each individual in the population is a labeling of all $|V|$ nodes, thus one individual occupies $O\big(|V|\big)$ space. At any generation, HP-MOCD holds both the parent population P_t and the offspring population Q_t. This amounts to $2N_p$ individuals, consuming

$$O\big(2\,N_p \cdot |V|\big) \;=\; O\big(N_p\,|V|\big) \quad \text{space.}$$

Regarding algorithmic components:

- For *Crossover*, a hash-map is employed to count labels dynamically. This map is reused for each node, resulting in $O(1)$ auxiliary space per node, which does not accumulate across nodes.
- For *Mutation*, apart from storing the new individual (requiring $O\big(|V|\big)$), a frequency map of size at most the node's degree ($d(v)$) may be allocated when mutating a single node. Since this map is reused across nodes, the total auxiliary space for this operation is $O(d_{\max})$.

The non-dominated sorting and crowding distance calculations (as in NSGA-II) are performed on $2N_p$ individuals. The space required for these operations includes:

$$\underbrace{O\big(N_p\big)}_{\substack{\text{domination counts}\\ \text{and front indices}}} \;+\; \underbrace{O\big(N_p\big)}_{\substack{\text{crowding}\\ \text{distance array}}} \;=\; O\big(N_p\big).$$

A.1 Total Space Complexity

The total space complexity is the sum of these components:

$$\begin{aligned}
&O\big(|V| + |E|\big) && \text{(graph storage)}\\
&+\, O\big(N_p\,|V|\big) && \text{(populations } P_t, Q_t\text{)}\\
&+\, O\big(d_{\max}\big) && \text{(mutation auxiliary space)}\\
&+\, O\big(N_p\big) && \text{(NSGA-II auxiliary space)}\\
&= O\big(N_p\,|V| + |V| + |E| + d_{\max} + N_p\big).
\end{aligned}$$

In typical scenarios involving sparse graphs (where $|E| = O\big(|V|\big)$) and for realistic algorithm parameters where $N_p\,|V|$ is the overwhelmingly largest term (i.e., $N_p\,|V| \gg |V|, |E|, d_{\max}, N_p$), the dominant space complexity is

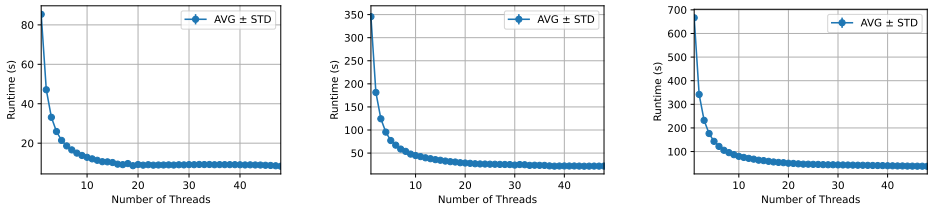
$$\boxed{O\big(N_p\,|V|\big).}$$

Appendix B Role of Multi-Threaded Architecture in Algorithm Performance

The inherent parallel design of our algorithm necessitates an investigation into its scalability with an increasing number of logical threads and the identification of optimal threading levels. Understanding this relationship is paramount to harnessing the algorithm's full potential.

To quantify the impact of multi-threading, we benchmarked the algorithm on a high-performance system featuring an AMD Ryzen Threadripper 3960X processor (24 physical cores, 48 logical threads, @ 3.70GHz) and 128GB DDR4 RAM. The number of execution threads was systematically varied from 1 to 48. These tests utilized the same synthetic graph datasets with $n \in [10,000, 30,000, 50,000]$ nodes as detailed in our scalability analysis (Section 4.1.1). Each configuration was executed five times, a sufficient number given the consistently low standard deviation observed in execution times.

Figure B1 illustrates the performance gains. Initially, increasing the number of threads yields substantial reductions in runtime. However, a performance plateau is reached beyond a certain threshold (approximately 8 threads for 10,000 nodes, 15 for 30,000 nodes, and 20 for 50,000 nodes). Beyond these points, adding more threads provides marginal benefits. This is likely attributable to the increased overhead associated with managing a large number of concurrent threads.



(a) 10,000 nodes (b) 30,000 nodes (c) 50,000 nodes

Fig. B1 Average runtime ($mean \pm \sigma$) as a function of the number of logical threads for graphs with 10,000 (a), 30,000 (b), and 50,000 (c) vertices. The results underscore the significant speedup achieved through multi-threading compared to single-threaded execution.

The most compelling evidence for the importance of the multi-threaded architecture lies in the stark contrast with single-threaded execution. As shown in Figure B1, a linear (single-thread) approach is dramatically slower across all scenarios. Specifically, when compared to utilizing the maximum available logical threads, single-threaded execution was **10.27** times slower for 10,000 nodes (Figure B1(a)), **16.45** for 30,000 nodes (Figure B1(b)), and **17.83** times for 50,000 nodes (Figure B1(c)). These figures unequivocally demonstrate that the algorithm's parallel processing capability is not merely an optimization but a fundamental requirement for achieving practical execution

times, rendering it viable and effective for real-world applications. Without its multi-threaded design, the algorithm would be impractically slow for datasets of meaningful scale.

Appendix C Selection of Genetic Algorithm Parameters

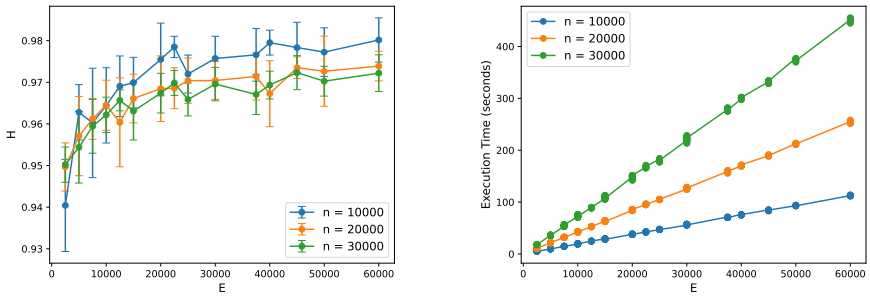
The performance of the NSGA-II algorithm depends critically on two GA parameters: the population size P and the number of generations G. These parameters control the total search effort and must be tuned to trade off clustering quality—measured here via AMI and NMI—against computational cost.

Aggregate Quality Metric

To summarize clustering quality in a single scalar, we compute the harmonic mean of AMI and NMI:

$$H \;=\; \frac{2\,\big(\mathrm{AMI}\times\mathrm{NMI}\big)}{\mathrm{AMI}+\mathrm{NMI}},$$

which balances the two scores and more strongly penalizes configurations where either AMI or NMI is low. While AMI and NMI are widely used, the methodology remains compatible with other external or internal evaluation metrics, depending on application context.



(a) H versus E (b) Execution Time versus E

Fig. C2 Harmonic mean of AMI and NMI versus GA effort $E = P \times G$, and execution time versus GA effort $E = P \times G$, for different graph sizes.

GA Effort Metric

As a proxy for total computational investment, we define:

$$E \;=\; P \times G,$$

i.e. the total number of candidate evaluations across all generations. This single metric captures both population diversity (P) and evolutionary depth (G).

A systematic grid search was performed over various (P, G) pairs for graph sizes $n \in [10\,000,\ 20\,000,\ 30\,000]$. For each configuration, we recorded H and the execution time.

Figure C2(a) and Figure C2(b) display, respectively, the harmonic-mean quality and runtime as functions of E, with separate curves for each n.

From these figures, we observe that clustering quality H increases rapidly with E up to approximately 10^4, after which gains become marginal. Meanwhile, execution time grows roughly linearly with E for all tested graph sizes. This behavior highlights a clear trade-off regime where further computational effort yields diminishing returns.

Chosen Configuration

To balance near-peak quality with reasonable runtime, we select:

$$P = 100, \quad G = 100, \quad (E = 10^4)$$

as this setting reaches near-optimal H before the saturation point, while keeping execution time moderate and scalable.