Towards Community Search in Uncertain Graphs

by

Yashar Talebirad

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science
University of Alberta

Abstract

The representation of real-world relationships and entities through nodes and edges in a network has found wide applicability across diverse scientific fields. At the core of network analysis are the tasks of community detection and community search, which aim to identify distinct groups within a graph. While community detection partitions the graph on a global scale, community search focuses on a specific node or group of nodes to discover a cohesive subgraph in their vicinity. Traditionally, these networks were represented as deterministic graphs with clearly defined nodes and edges. However, as networks grow in scale, analyzing these networks becomes more challenging. Coupled with this, the emergence of uncertainty in data collection has necessitated a shift towards probabilistic modeling of these relationships, presenting a suite of new complexities and challenges. In response to these challenges, this thesis first focuses on enhancing the SIWO algorithm, initially designed for community mining in deterministic graphs, to make it suitable for processing very large graphs. We introduce a methodology to convert large graphs into a format that is more manageable by local community search algorithms, ensuring efficient processing without the need to store entire networks in main memory. This is complemented by the development of data structures and optimization techniques specifically designed to manage and process large-scale network data efficiently. Building upon these enhancements, we then present USIWO, a scalable and local algorithm for community search in unweighted uncertain graphs with edge uncertainty. USIWO starts from a single node or set of nodes and incrementally adds "suitable" adjacent nodes one at a time, until it rapidly finds the core of a community even in very large uncertain networks.

Preface

The work done in Chapter 3 of this thesis has been published under the title "Fast local community discovery relying on the strength of links" at the Social Network Analysis and Mining (SNAM) 2023 journal [1]. The work done in Chapter 4 has been accepted for publication under the title "USIWO: A Local Community Search Algorithm for Uncertain Graphs" at the ASONAM 2023 Conference.

Dedicated to raising awareness of mental health, in the hope that it may contribute to greater understanding, support, and compassion for those affected.

Acknowledgments

I want to express my gratitude to my supervisors, Osmar R. Zaiane and Christine Largeron, for their kind help and advice. Without their help, this work would not have been possible. I also want to give my thanks to my teammates in Meerkat, who helped me in my research.

Table of Contents

List of Tables

List of Figures

Abbreviations

ACR Average clustering reliability.

AVPR Average vertex pairwise reliability.

CTC Closest Truss Community.

DBCLPG Density-Based Clustering of Large Probabilistic Graphs.

LCTC Local CTC.

LFR Lancichinetti–Fortunato–Radicchi benchmark.

LTE Local Tightness Expansion.

NMI Normalized Mutual Information.

PPI Protein-Protein Interaction.

PWF Pairwise F-Measure.

SIWO Strong In, Weak Out.

SNA Social Network Analysis.

TCE Triangle-Based Community Expansion.

URGE Uncertain Graph Embedding.

USIWO Uncertain SIWO.

Chapter 1

Introduction

1.1 Motivation and Background

In recent years, networks have become essential tools for modeling complex systems across a variety of domains such as social sciences, biology, pharmacology, criminology, and computer science. They provide a framework for representing relational data through graphs, where vertices (or nodes) symbolize entities and edges (or links) correspond to the relationships between these entities. As emphasized in "The Future is Big Graphs" [2], the sudden growth in the amount of interconnected data highlights the importance of graph processing in modern society. With this growth, however, comes significant challenges in analyzing networks that have expanded to encompass millions of nodes and billions of edges. Effectively handling and understanding these large networks necessitates the development of new methods and scalable algorithms, which need to be specially designed to deal with the challenges of handling large and complex network structures. Additionally, while commonly these networks have been modeled as deterministic graphs, the reality of many real-world scenarios, influenced by factors such as data collection processes and inherent relationship ambiguities, often introduces an element of uncertainty [3]. This uncertainty has led to the development of "Uncertain Graphs" or "Probabilistic Graphs", which embed the uncertainty directly into the graph's structure [4]. Such graphs can represent various sources of uncertainty, whether in edges, nodes, or attributes. Particularly, uncertain

graphs with uncertainty on edges are often used to represent noisy connections between data, such as in Protein-Protein Interaction (PPI) networks, social media, and brain activity representation [5]. Additionally, these graphs are also used in scenarios where the relationships between nodes are not explicitly known and need to be predicted, which is particularly seen in domains such as recommendation systems or link prediction in social networks [6].

As networks continue to grow in size and complexity, two tasks—community detection and community search— have been getting more attention. These tasks focus on finding the inherent structure within networks characterized by communities, which are groups of vertices that are densely interconnected within the network yet sparsely connected to the rest of the network [7]. While community detection aims for a comprehensive partitioning of all graph vertices, community search offers a more localized approach, focusing on specific nodes or node groups to find densely-connected subgraphs containing those nodes. Such a local approach, especially on large graphs, stands as a promising solution for detecting communities in large uncertain networks. This kind of approach is especially useful for cases where the entire graph structure may be too large to store in main memory. Furthermore, the shift from deterministic to uncertain graphs demands new algorithms, as naive approaches often fall short in being accurate [8].

In light of these challenges, this thesis first addresses the enhancement of the SIWO algorithm for performing community search in large-scale networks. This algorithm is an extension of the *SIWO* method initially described in [9] for global community detection. Following this, we present USIWO as a community search algorithm designed for large-scale unweighted uncertain graphs with independent edge probabilities.

1.2 Problem Definition and Challenges

The primary objective of this thesis is to address two challenges in network analysis: to effectively manage the complexities associated with large-scale networks, and to

address the challenges of community search in uncertain graphs. Given an uncertain graph $G = (V, E, p)$ where V represents the set of nodes, E the set of edges, and p the probabilities of those edges, our objective is to identify meaningful communities around a given node or set of nodes. The inherent uncertainty in these graphs, coupled with their potentially massive size, poses significant challenges. Traditional community mining algorithms, both global and local, are primarily designed for deterministic networks, which are networks in which the connections have been observed with complete certainty. Applying these algorithms to uncertain networks without modifications can lead to inaccurate community structures. Furthermore, the sheer scale of large graphs, often featuring millions of nodes and billions of edges, presents an additional layer of complexity. These large graphs necessitate algorithms that can efficiently process vast amounts of data without compromising accuracy. The desired properties of a suitable community search algorithm for uncertain graphs, therefore, include:

- **Handling edge uncertainties:** Determining the strength of relationships between nodes when edges are probabilistic.
- **Scalability:** Efficiently processing large-scale uncertain networks without compromising accuracy.
- **Real-Time Implementation:** Being able to find communities in real-time or a limited amount of time by only traversing the required parts of the network.
- **High Accuracy:** For a query node, outputting a community with the highest possible number of nodes from its true community without including outliers.

Addressing these challenges requires an approach that balances the complexities of uncertain graph structures with the practical demands of processing large networks. The development of such methods is a key focus in this thesis.

1.3 Thesis Statements

This thesis explores the field of community search in large uncertain networks, with a particular focus on the SIWO algorithm and its adaptations for such scenarios. The primary hypotheses explored are:

- **Thesis Statement 1:** Specialized data structures and optimizations can be employed to enhance the efficiency of reading and processing large input files. These improvements are particularly effective for community search when integrated into the SIWO algorithm.
- **Thesis Statement 2:** The SIWO algorithm, designed for community mining and search in a graph, can be modified in a way that allows for flexible execution, ensuring partial (but accurate) outputs even within specific time constraints.
- **Thesis Statement 3:** Uncertain networks carry probability values within their edges. By effectively utilizing these probabilities, one can quantify how strong these edges are, rather than merely recognizing them as weights, thus refining the process of community detection and search.
- **Thesis Statement 4:** The SIWO algorithm can be adapted and optimized for uncertain networks to yield a coherent and meaningful community structure containing the query node(s).

1.4 Thesis Contribution

The primary contributions of this work include:

1. Introduced a methodology to convert large graphs to a more manageable form, enhancing their suitability for local community search algorithms. This transformation ensures that large-scale networks are efficiently processed, without the need to store them in main memory. This is explored in Section 3.1.1.

2. Enhanced SIWO by utilizing data structures and optimization techniques designed for handling large input files. These methods significantly improve the efficiency of reading and processing data, making subsequent analysis on large networks possible. This is explored in Sections 3.1.2 and 3.1.3.

3. Introduced a flexible implementation[1] of the enhanced SIWO algorithm that allows it to operate within given time constraints, producing accurate partial community detection results even when faced with limited time. This is explored in Section 3.1.4.

4. Conducted several experiments to showcase the scalability of the enhanced SIWO algorithm on large synthetic and real-world networks. This is detailed in Section 3.2.

5. Extended and optimized the enhanced SIWO algorithm for uncertain networks, introducing the extended version as the USIWO algorithm. This adaptation ensures that USIWO is an accurate algorithm capable of finding coherent community structures when faced with probabilistic networks. This is explored in Sections 4.1 and 4.2.

6. Devised an uncertain network generator that converts deterministic graphs to uncertain graphs, introducing varying degrees of uncertainty while preserving the community structure. This is detailed in Section 4.3.1.

7. Conducted a variety of experiments to showcase the performance of the USIWO algorithm on synthetic and real-world networks that have undergone several variations of uncertainty, and compared them with several contenders. This is covered in Section 4.3.4.

[1]Code available in Github.

1.5 Thesis Organization

This thesis is organized into several chapters, each focusing on a distinct aspect of our study. The following outlines the structure and key focus of each remaining chapter.

Chapter 2 provides a comprehensive review of existing community detection and search algorithms, detailing their strengths and limitations. A review of social network analysis, community detection, community search, as well as the common evaluation methods used in the literature, is also provided. We finish the chapter by reviewing SIWO, a community mining algorithm that we build upon.

In Chapter 3 we describe the challenges posed by large graphs and present the ways we adapted SIWO to handle them. We introduce a method to convert large input graphs to a format that is more compatible for local community search algorithms. We then detail specialized data structures and optimization techniques that facilitate the efficient reading and processing of vast input files. The chapter concludes with the experiments done with SIWO on large networks.

Chapter 4 highlights the transition from deterministic to uncertain networks. It begins with an exploration of the complexities introduced by the probabilistic nature of edges in uncertain networks. The chapter then presents the *USIWO* algorithm, an adaptation of the SIWO algorithm, designed to utilize the probabilities on edges to calculate their strengths in order to perform community search. The chapter concludes with the experimental results that demonstrate the capabilities of USIWO.

The thesis concludes with Chapter 5, which offers a summary of the research contributions and discusses potential future work.

Chapter 2

Background and Related Work

In this chapter, we cover important works related to our research, mainly in the areas of social network analysis and community detection. We begin by introducing and explaining key concepts and terms essential for understanding the subsequent discussions. After establishing the foundational knowledge, we discuss various metrics used in the relevant studies, ranging from standard measures such as accuracy to those used specifically for uncertain graphs. We then take a close look at the relevant literature and existing methodologies focused on uncertain graphs with an emphasis on two key areas: Community Detection, and Community Search. The last section of this chapter is dedicated to the SIWO algorithm [1]. SIWO is a flexible algorithm that can be updated to a global algorithm which can be run on the whole input graph. It has also been improved to take weighted graphs into account, which has also been shown to outperform its competitors [10]. In this thesis, we refine SIWO by introducing a new version capable of handling graphs of any size, as detailed in Chapter 3. Additionally, we extend its functionality to work on uncertain graphs, which is detailed in Chapter 4.

2.1 Preliminary Definitions

We first explain some key terms that is necessary to understand our contribution to the field.

A network in its simplest form is represented by a graph $G = (V, E)$ where V is the set of n nodes ($|V| = n$) and E is the set of m edges ($|E| = m$)

Node (or Vertex) A discrete entity or point in the network. In a social context, it can represent an individual, organization, or any other entity.

Edge (or Link) Represents a relationship or connection between two nodes. In uncertain networks, this connection is often associated with a probability, indicating the likelihood of its existence.

Degree The number of edges connected to a node. For uncertain networks, this can be represented as an expected degree, which incorporates the probabilities of the edges to account for uncertainty in connections.

Adjacency Matrix A square matrix used to represent a finite graph. The elements indicate whether pairs of vertices are adjacent or not in the graph.

Probabilistic Adjacency Matrix In uncertain graphs, the adjacency matrix is extended to capture the uncertainty associated with each edge. Instead of binary entries, each element in the matrix represents the probability of the existence of an edge between respective vertices. An entry of 0 indicates no possibility of an edge, while a value of 1 would mean the edge's existence is certain, and values between 0 and 1 indicate varying degrees of uncertainty.

Communities (a.k.a Clusters) Community structure in a network indicates the presence of groups of vertices, where intra-group edge density is higher than inter-group edge density. These densely-connected vertex groups are termed communities [11].

Realization (a.k.a. Possible World) A specific deterministic network derived from an uncertain one when considering the existence of a particular set of probabilistic edges.

Subgraph A graph formed from a subset of the nodes (and respective edges) of a larger graph.

Connected Component A subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph.

2.2 Social Network Analysis

Social Network Analysis (SNA) refers to a set of methods used to analyze social networks, which are structures determined by relationships and entities. Originating from sociology and anthropology, it has over the years been applied to diverse fields, including computer science, epidemiology, and organizational studies [12]. In social network theory, a network is represented as a graph, where nodes represent entities (e.g., people, organizations) and edges denote relationships (e.g., friendships, collaborations). In social network analysis (SNA), the emphasis is placed on the inter-connectedness and inter-dependencies that exist among the nodes [13].

2.2.1 Deterministic Networks

Deterministic networks have clear and straightforward relationships between nodes, where an edge connecting two nodes represents an unambiguous relationship. This clarity makes it easier to understand and analyze the network [14].

Deterministic graphs have evolved to include more complex forms, such as weighted and attributed graphs. In weighted graphs, edges are assigned values that represent specific characteristics of the relationship between the nodes [15]. Attributed graphs, extend this concept by incorporating additional data, primarily in the form of attributes. These attributes are typically associated with nodes, and provide more contextual information about the entities [16].

Though the relationships between nodes in a deterministic graph are clear, the

networks themselves can change over time. Some networks are static, showing relationships at a specific moment, while others are dynamic, reflecting the flow of connections as they evolve. These networks are also known as Temporal Networks [17].

The unambiguous nature of deterministic networks enables the use of various analytical metrics. For instance, centrality measures are used in revealing the most influential or central nodes within a network [12]. Furthermore, deterministic networks find applications in many areas. Sociograms, for example, are visual charts that describe social relationships [18]. Moreover, in both academic and corporate settings, these networks can effectively map out collaborations, highlighting the professional interactions among collaborators [19].

2.2.2 Uncertain Networks

While deterministic networks have been widely studied in the past, uncertain networks are now getting more attention as they represent the complex and often uncertain nature of today's world. This is because traditional networks, where relationships between nodes are deterministic, often fall short when modelling real-world phenomena where uncertainty is inherent. In fact, numerous real-world networks are characterized by uncertainty, which can arise from various sources such as the data collection process or pre-processing methods that use machine learning [3]. This has led to the development of "Uncertain Networks" also known as "Probabilistic Networks", which incorporate uncertainty into the network structure.

In these networks, the existence of an edge does not guarantee a relationship; it just indicates a possibility. Every connection or edge in such a network is tagged with a probability, indicating the likelihood of its existence. Because of this uncertainty, an uncertain network can lead to many possible deterministic networks (known as possible worlds), each showing a different set of relationships based on the probabilities. This variety brings both challenges and new ways to understand these types of

networks.

Uncertain networks can also be seen in real-life situations. For instance, in online communication, they can be used to map out the spread of information, especially when the relationships or connections have inherent uncertainty, such as in the spread of rumors or unverified news. In online platforms, networks might represent interactions with associated trust scores, reflecting varying levels of confidence in different interactions. In bioinformatics, they can show possible interactions when certain biological pathways are based on approximate statistical models [20].

However, the probabilistic nature of these networks brings forth its own set of challenges. Computationally, managing and analyzing these networks, especially when considering the large number of possible realizations can be infeasible. Furthermore, the task of gathering accurate probabilistic data for relationships can be both resource-intensive and challenging.

Generally speaking, Uncertain Graphs are graphs with any type of uncertainty. In an uncertain graph, uncertainty can be associated with one or several of the following components:

- **Edges**: This is the most common form of uncertain graphs, in which each edge is associated with a probability, with various interpretations based on context.
- **Nodes**: In this case, a probability value is assigned to each possible scenario based on the node states. Many of the problems that are studied on graphs with edge uncertainty can also be studied on graphs with node uncertainty.
- **Attributes**: More complicated forms of uncertainty can arise when the graph contains attributes on nodes, edges, or both. The uncertainty would then cause the attributes to change in different scenarios.

Uncertain graphs with edge uncertainty help deal with most situations where the node connections are probabilistic, and are frequently used to represent noisy connections between data. PPI (protein-protein interaction) networks are a well-known

example of this type, with nodes representing proteins and edges representing the interactions between them. In this case, the edge probabilities are the likelihoods of them existing [20]. Other use cases of uncertain graphs with edge uncertainty include social networks, where probabilities can represent the likelihood that two individuals know each other or share a similar interest, and machine learning, where probabilities describe correlations between events. These graphs are also used in neuroscience, for instance, to represent activity in different brain areas, as discussed in a case-study by [5].

In uncertain graphs with edge uncertainty, a common approach in research is to treat the probabilities on the edges as independent. This means that the existence of one edge does not influence the probability of another edge in the graph. These graphs are usually interpreted using the possible-world semantic, where each edge is sampled independently according to its assigned probability. This model of uncertainty does not account for the relationships between the edges but remains the most prevalent approach due to its simplicity and effectiveness in various applications. Therefore, it is used as the primary model for uncertain graphs in this thesis.

It is important to note that dependencies between edge probabilities can also exist, which adds another layer of complexity to the analysis and interpretation of these graphs. For instance, work such as that presented in [21] explores clustering methods in uncertain graphs with correlated probabilities. However, due to the increased complexity and less common usage in current research, the correlated probabilities model is not further explored in this thesis. Following the conventional model of probabilities, the simplest and most widely used definition of uncertain graphs, introduced in [22], is akin to:

Definition 1 *An uncertain graph $\mathcal{G} = (V, E, p)$ where $p : E \to (0, 1]$ can be viewed as a probability space whose outcomes are sub-graphs $G' = (V, E')$ of $\mathcal{G}$ where any edge $e \in E$ is included in E' with probability $p(e)$ independently. These possible outcomes*

are also referred to as "Possible Worlds", each representing a potential realization of the uncertain graph [23].

The problem of mining "Attributed" uncertain graphs is also studied in [24]. These graphs are represented as $\mathcal{G} = (V, E, F, p)$, where F is a set of n attribute vectors indicating the d attributes associated with each node, and p maps every pair of nodes to a real number in $[0; 1]$.

Following these definitions, several problems can be studied in the subject of Uncertain Graphs. These problems typically involve taking a problem of interest (especially mining problems) on deterministic graphs, proposing a relevant question on uncertain graphs using new definitions, and finally, finding new algorithms to solve the newly-defined problem. Examples of such problems include (but are not limited to) computing several properties between nodes (e.g. Reliability), mining problems (e.g. Clustering, Link Predictions, Node Classification), and graph algorithmic problems (e.g. Most Reliable Spanning Tree, Information Flow Maximization). A thorough analysis of these problems on probabilistic graphs is done in [25].

Although these issues have been widely researched on deterministic graphs, transitioning to uncertain graphs introduces new complexities, as it necessitates the redefinition of existing problems and the development of novel algorithms. To mention the most important challenge in this transition, note that since there is a probability associated with the existence of some edges (with size m), the resulting deterministic network generated by the probabilistic graph can take 2^m different forms, which can be exponentially difficult to deal with. Thus, doing an exhaustive search on all the possible worlds would be infeasible. In this thesis, we focus on the problem of community search in uncertain and deterministic networks.

2.3 Community Detection and Search

In networks, clusters of interconnected nodes within larger networks are known as community structures. These are the groups of vertices that are densely connected within the network and sparsely connected with the rest of the network [7].

Traditionally, community detection methods aim to find these structures by globally partitioning a network and identifying all possible communities through the analysis of various network features and properties. In this thesis, the terms "community detection", "community mining", and "clustering" are used interchangeably and convey the same meaning. Various methods are used as potential solutions to this problem. For instance, direct edge densities within a potential community can be utilized to identify "bridges" that separate communities, or the information flow within the network can be evaluated to assist with community detection [26].

On the other hand, as noted in "The future is Big Graphs" [2], the current era of big data and notably large networks have brought local search methods into the spotlight. This is because the necessity to store and process all network information, especially in scenarios involving networks with a large number of vertices and edges, makes global approaches increasingly impractical and resource-intensive.

Local community discovery or community search approaches are more localized, initiating a search from a specific query node or set of nodes [27, 28]. These methods aim to identify nodes that belong to the same community as the query node(s), without the need to explore and analyze the entire network. This strategy not only reduces computational time, making it applicable even in large networks, but also proves beneficial in applications where the interest is primarily in the community of a specific entity. Furthermore, local methods offer a practical solution for online searches and managing dynamic graphs that change over time [29].

Studies that focus on community search or community detection also tend to consider different metrics to measure the accuracy of the output of their proposed algo-

rithms. We will now go over a number of the metrics that have been used by the related literature.

2.4 Evaluation Metrics

There are several ways to determine how good a community detection or search algorithm is. These methods are often very different in terms of the existence or non-existence of "ground truth" communities. Ground truth communities refer to pre-established and accepted sets of nodes known to form a community. If these known communities are trustworthy, we can compare the communities found by the algorithm to the known ones to see how good the algorithm performs. If the known communities are not trustworthy or non-existent, other measures can be used for evaluation.

2.4.1 With Ground Truth

In scenarios where the community structure within a graph is well-documented and reliable, several metrics are used to evaluate the performance of community detection and search algorithms.

For community search, various metrics compare the discovered community of a specific node to the node's established ground truth community. We will now formally define several of these metrics, which are applicable to a specific node or set of nodes belonging to a community.

Definition 2 (Precision) *: Precision determines the fraction of vertices in the detected community c that belong to the actual community g.*

$$Precision(c, g) = \frac{|c \cap g|}{|c|}$$

Definition 3 (Recall) *: Recall represents the fraction of vertices in g that the algorithm correctly identifies within c.*

$$Recall(c, g) = \frac{|c \cap g|}{|g|}$$

In the literature, precision and recall are often combined into the F score (also called the $F_1 - Score$, or simply F_1) as a more comprehensive metric.

Definition 4 (F Score) *: The F Score provides a balanced measure, combining both precision and recall.*

$$F(c, g) = \frac{2 \times Recall(c, g) \times Precision(c, g)}{Recall(c, g) + Precision(c, g)}$$

The values of precision, recall, and F score lie between 0 and 1. Higher values indicate a more accurate match with the true community.

All three of these metrics (precision, recall, and F score) can also be applied to assess community mining methods that are designed for uncertain graphs. In uncertain graphs, the discovered community for a node is still a set of nodes, even if they are determined under the influence of edge uncertainties. This makes it feasible to compare this discovered set with a known ground truth set using precision, recall, and F score. Essentially, despite the probabilistic nature of uncertain graphs, the end goal remains consistent: identifying a community that most closely matches the true community.

In situations where community detection algorithms are applied, and the entire detected community structure is compared against all known communities, the Normalized Mutual Information (NMI) metric can be used. NMI is a metric used to measure how much information one set shares with the other. In community detection, NMI measures the similarity between two sets of clusters or communities, by evaluating the correspondence between the discovered community structure and the known ground truth.

Definition 5 (Normalized Mutual Information (NMI)) *:*

$$NMI(X, Y) = \frac{2 \times I(X; Y)}{H(X) + H(Y)}$$

Here, $I(X; Y)$ represents the mutual information between sets X and Y, and $H(X)$ and $H(Y)$ are the entropies of X and Y respectively.

The metric yields a value between 0 and 1, where a higher value indicates a greater similarity between the two community structures. Specifically, an NMI value of 0 suggests no mutual information, implying the two sets of communities are entirely dissimilar, whereas a value of 1 means the sets are identical.

Other variations of NMI exist that use different normalizing factors, such as the maximum, the sum, the square root, or the minimum of the marginal entropies of the two different clusterings [30].

2.4.2 Without Ground Truth

In many scenarios, the ground truth for the community structure within a graph is either unreliable or unknown, making it challenging to evaluate the quality of community detection algorithms. Under these circumstances, alternative metrics can be utilized to assess the effectiveness of these algorithms in identifying meaningful communities.

Various methods are available for evaluating community detection in deterministic graphs, such as Modularity [31]. Miasnikof et al. [32] provide different metrics for analyzing graph clustering algorithms. However, these metrics are specifically used for measuring the performance of clustering algorithms in deterministic graphs, and thus, extra measures must be taken when applying them to algorithms designed for uncertain graphs. Some of these techniques are explained in [5]. In this section, we mainly focus on detailing the evaluation metrics specifically designed to assess algorithms operating on uncertain graphs.

One fundamental concept underlying many of these metrics is "Reliability" [33]. In a connected deterministic graph, by definition, all pairs of nodes are interconnected. This concept, however, requires adaptation for probabilistic graphs, where connections between nodes are not guaranteed but are instead characterized by probabilities. Reliability extends this concept to uncertain graphs by quantifying the likelihood that a set of vertices remains interconnected across different deterministic

instances of the graph, known as "possible worlds". We now proceed to formally define reliability.

Definition 6 (Reliability) *Given an uncertain graph $\mathcal{G} = (V, E, p)$ and a subset of vertices $V_s \subseteq V$, the reliability $R(V_s)$ for the vertex set V_s is defined as:*

$$R(V_s) = \sum_{G_i \sqsubseteq \mathcal{G}} \Pr(G_i) I(V_s, G_i), \tag{2.1}$$

Where G_i is a possible world of $\mathcal{G}$, and $I(V_s, G_i)$ is the indicator function of whether V_s is contained in a connected component in G_i (in which case it is set to 1), or not (in which case it is set to zero).

Note that the reliability value lies in the interval $[0, 1]$, whereas deterministic connectivity is a binary value. This measure was later used by researchers to devise metrics to quantify how good a clustering algorithm is. For instance, in [34] the authors first generalize the concept of reliability even more by utilizing information theory, coming up with a new formula. Later, they use the metrics ACR and AVPR to compare their clustering algorithm to other contenders in the literature. These metrics are defined as follows:

Definition 7 (Average clustering reliability (ACR)) *: ACR is defined as*

$$ACR(C) = \frac{\sum_{i=1}^{k} |C_i| R(C_i)}{n} \tag{2.2}$$

Where $C = \{C_1, C_2, \cdots, C_k\}$ is the set of k clusters in an uncertain graph with n nodes, and R is the reliability measure.

ACR quantifies the probability of the nodes remaining connected in each cluster and is used to measure the reliability of each cluster obtained by a clustering algorithm. A high ACR means that the clusters are likely to be connected subgraphs in most possible worlds, and a low ACR means that the clusters are likely to be broken into fragments in many possible worlds. ACR is designed to penalize clusters that are not robust to edge uncertainty.

Definition 8 (Average vertex pairwise reliability (AVPR)) *: AVPR is defined as:*

$$AVPR(C) = \frac{2\sum_{i=1}^{k}\sum_{u,v\in C_k} R(\{u,v\})}{\sum_{i=1}^{k}(|C_i|)(|C_i|-1)} \tag{2.3}$$

AVPR is the average connection probability of all pairs of nodes that are in the same cluster (inner) and different clusters (outer). Definition 8 is for inner-AVPR, which is referred to as simply "AVPR" in some literature. A high AVPR means that the nodes in the same cluster are likely to be connected in most possible worlds, and a low AVPR means that the nodes in the same cluster are likely to be disconnected in many possible worlds. AVPR is designed to penalize clusters that contain nodes that are not well-connected.

ACR and AVPR are later used by other works as measures for the comparison of clustering algorithms [24, 35].

To evaluate the accuracy of the different clustering algorithms, Li et al. [24] use the generalized metrics of accuracy for clustering, such as Accuracy (ACC) [36] and Pairwise F-Measure (PWF) [37]. For clustering algorithms that are center-based, there are additional metrics that can be used (as in [35]):

- p_{avg} : Average connection probability of all nodes to their respective centers
- p_{min} : Minimum connection probability of any node to its center

In some literature ([38] and [5]), Jaccard index (or coefficient) [39] is used for performance evaluation. The Jaccard index between two sets is defined as the size of the intersection of the sets divided by the size of the union. In the subject of graph clustering [5], this metric is used to measure the similarity between the clusters. However, Hu et al. [38] use an expected form of the Jaccard index as a similarity measure between nodes.

2.5 Literature Review

In this section, we provide a survey of the existing body of work, focusing primarily on community detection and community search within uncertain networks. We also briefly mention related works on community detection and search within deterministic networks. Although Community Detection and Community Search are different in nature, there is a strong parallel between them since one could perform partitioning by iteratively applying a community search algorithm on a query node, removing the found community from the network, and repeating this process until the entire network is partitioned into communities. Each community detected by the algorithm corresponds to a cluster in the network.

Most community mining and search algorithms work towards optimizing a certain objective, to find the "best" communities among the nodes. Most of these algorithms use one or both of the following definitions:

1. Defining how "similar" a pair of nodes are in the network.

2. Defining a function to measure how "good" a community is.

For instance, Modularity is a concept that captures how well a network can be divided into groups of nodes that are densely connected internally and sparsely connected externally [31]. It is based on the idea that in a network, the deviation from randomness can indicate the presence of communities, as random network would not have such a clear community structure. However, modularity also faces some limitations, such as the resolution limit, which prevents it from detecting small communities in large networks [40], and the inconsistency problem, which makes it unreliable for finding statistically significant communities [41]. Therefore, modularity should be used with caution and complemented with other methods and criteria.

In modularity-based algorithms, modularity is used as an objective function to optimize in order to find the best partition of a network into communities. By max-

imizing modularity, the groups of nodes that have more internal connections than expected by chance can be identified [42]. For instance, Modularity Q [43], Modularity R [44], Modularity M [45], and Modularity L (or L-Metric) [46] are among these algorithms. Although these algorithms are devised for deterministic networks, they can be made to work for probabilistic graphs by extending the definition of modularity to account for all possible worlds. For instance in [47], the authors have extended the definition of Modularity R and followed a greedy approach to find the best community.

2.5.1 Community Detection in Uncertain Networks

To the best of our knowledge, Liu et al. [34] were the first to have studied and formalized the definition of clustering for uncertain graphs. The formalization of the problem made way for other researchers to study the newly-formed problem.

In this section, we are going to review the contributions of the literature surrounding the problem of community detection in uncertain graphs, and discuss their shortcomings. The algorithms that are proposed for the uncertain graph clustering problem can be divided in several categories, based on their main approach:

1. Information Theoretic Approach
2. Edit-Distance-Based Approach
3. Similarity-Based Approach
4. Connectivity-Based Approach

It is worth noting that both edit-distance-based and connectivity-based approaches are based on the notion of "center", i.e. they rely on identifying a center for each cluster. However, their objective is entirely different. Each of the following sections is dedicated to one of these categories.

Information-Theoretic Approach

In 2012, Liu et al. [34] formalized the definition of clustering for uncertain graphs and proposed new metrics to measure how good a clustering is. The paper's main contribution revolves around a generalization of "Reliability" in probabilistic graphs, which evaluates the connectivity of a set of nodes in the context of the entire network, using the possible worlds model. Generalized reliability is based on the intuition that reliable clusters are clusters that are not likely to be disconnected in different possible worlds.

This measure is then used to develop a novel k-means algorithm for clustering using information theory. This generalization relies on the two measures "Purity" and "Size Balance" that are intuitively helpful to differentiate a good clustering from an inaccurate one.

Since each possible world may break the graph into several connected component (fragments), given a possible world, if two nodes are from the same underlying cluster then they are expected to be contained in the same fragment. This is the basis of the Purity measure: For each fragment of a possible world resulting from uncertain graph sampling, the number of distinct clusters to which the different nodes belong should be as small as possible, and one of the clusters should dominate the fragment. This measure is then defined with the help of cluster label entropy. However, the minimization of purity tends to bias the process towards unbalanced clusters. In other words, if the purity criterion is used alone, the result will typically contain a single cluster containing most of the nodes. To mitigate this, "Size Balance" is added to the clustering criterion, which measures how balanced the clusters are, in terms of the number of nodes in each.

Using these definitions, an objective function is created which incorporates both size balance and purity. To devise an algorithm that aims to maximize that objective function, an Auxiliary Cluster Table is created for each possible world. In this table,

each row corresponds to a cluster, each column corresponds to a connected component, and each cell contains the set of vertices from the corresponding component and cluster. The Auxiliary Cluster Table not only provides a structured view of the possible world but also contains all the information for the clustering, including the cluster and fragment identifier for each vertex. Then, Hoffman Encoding is used to encode each table. It is then shown that the coding length for the tables is directly related to the aforementioned objective function. This relation is utilized to devise a two-step procedure aimed at minimizing the objective function for the N sampled graphs. This procedure assigns each vertex randomly to one of the clusters, and repeats the following two stages until the objective converges:

- **Coding Computation**: Given the current vertex assignment, and corresponding sampled components, the coding for each row of the table is computed using Huffman coding.

- **Vertex Assignment**: Given the current row coding for each table, one can assign each vertex to the cluster that minimizes the total coding cost for that vertex through all the sampled graphs.

It can be shown that this procedure makes the objective function converge to a local optimal value. Finally, another algorithm is used on the resulting clustering to ensure connectivity in each cluster. The resulting algorithm is named "coded k-means". Finally, experimental studies are conducted to demonstrate the effectiveness and efficiency of the algorithms.

Two groups of "effectiveness" experiment results are reported:

1. **Generalized reliability** (the objective function): To compare the accuracy through different datasets, the average coding length per vertex is used.

2. **Standard reliability**: Acts as a more objective measurement for the results. Includes two measures: Average vertex pairwise reliability (AVPR) which em-

phasizes the pairwise reliability, and average cluster reliability (ACR), which puts a stronger constraint that requires all vertices in one cluster should be connected simultaneously.

The contenders that are used for comparison are MCL [48], Spectral Clustering, and Ensemble Clustering [36], which are methods used for deterministic graphs. In order to use these algorithms, the authors have converted the uncertain graphs into weighted graphs. It is then shown that the coded k-means algorithm produces better clusters than the aforementioned previous works on both the generalized and standard reliability criteria.

To compare scalability, databases with 20k to 100k nodes were tested, to show that the running time of the coded k-means algorithm is linearly related to the number of nodes. In short, the use of information theory in this paper in the context of graph clustering is novel, which has led to the definition of a new measurement for comparing clustering algorithms. However, there are some shortcomings to this approach. First, the intuition behind "Size Balance" is not accurate for small-world graphs, because these graphs feature a limited number of very large clusters along with numerous very small clusters. Also, the number of clusters is a parameter that is considered a feature, which is fixed and fine-tuned when compared to other algorithms. Lastly, the contenders are designed for weighted networks, not uncertain networks.

Edit-Distance-Based Approach

In 2013, Kollios et al. [49] approach the problem of clustering uncertain graphs in an entirely different way, through a concept called "edit distance". The edit distance between two graphs (on the same set of nodes) is defined as the number of edges that needs to be added and removed from one graph to get the other graph. This concept was already used in previous literature, in which a "clustering" is viewed as a cluster graph which is basically a graph consisting of disconnected cliques, to provide a new definition of graph clustering for deterministic graphs. The ClusterEdit problem was

first introduced by Shamir et al. [50] for deterministic graphs. In this problem, the goal is to find the cluster graph that has the minimum edit distance from the input graph.

The authors extend this edit-distance-based definition of graph clustering to probabilistic graphs. In this paper, pClusterEdit is the problem of finding the cluster graph that has the minimum **expected** edit distance from the input graph, making it a generalization of the ClusterEdit problem. This objective function is parameter-free, which means that the number of clusters does not need to be known in advance. At first, a connection between the objective function and the correlation clustering problem is exploited to propose practical approximation algorithms for the clustering problem.

The four algorithms that were used for the pClusterEdit problem are:

1. **The pKwikCluster algorithm**: In the correlation clustering problem, the goal is to group vertices according to their similarity to each other. This similarity is represented in a binary manner, by "0" or "-" meaning dissimilarity, and "1" or "+" meaning similarity. In probabilistic graphs, the probability on the edges can be seen as the likelihood that an edge is labeled "+". However, this setting requires the input graph to be complete, which means no pair of nodes has the edge probability of zero.

   Ailon et al. [51] proposed the KwikCluster algorithm for the weighted Correlation Clustering problem. Kollios et al. utilized this algorithm for probabilistic graphs. In the pKwikCluster algorithm, a random node u is selected, and a cluster with all neighbors of u that are connected with u with a probability higher than $\frac{1}{2}$ is created. If no such node exists, u defines a singleton cluster. After removing u and its cluster neighbors, the algorithm proceeds with the rest of the graph. When the clustering is completed, the result is a set of disconnected cliques. This algorithm is called pKwikCluster, which is a randomized

expected 5-approximation algorithm for the pClusterEdit problem.

The worst-case time complexity of the pKwikCluster is $O(m)$, m being the number of edges, which makes it linear with respect to the size of the input.

2. **The Furthest algorithm**: The Furthest algorithm uses the notion of centers for a top-down approach. At each step, the node that is not a center and has the maximum distance (i.e. minimum probability) to the current centers is selected as the new center. The distance of a node from the current centers is defined as the maximum among the probabilities of having an edge with the current centers. The nodes are then assigned to the center to which they are connected with maximum probability. At the end of each iteration, if the edit distance between the current cluster graph and the input is less than that of the last iteration, the algorithm continues to the next iteration. Otherwise, it stops and outputs the previous cluster graph.

   The Furthest algorithm needs to compute the distance between every node with the selected cluster centers. If k clusters are formed in the end, the worst-case running time of the Furthest algorithm is $O(n \cdot k^2)$.

3. **The Agglomerative algorithm**: The Agglomerative algorithm is a bottom-up procedure, which can be seen as the inverse of the Furthest algorithm. It starts with all nodes in a singleton cluster, then finds the pair of clusters with the highest average probability and combines them. The algorithm ends when the highest average probability is greater than $\frac{1}{2}$.

   A naive implementation of the Agglomerative algorithm requires $O(k \cdot n^2)$ time, where k is the number of clusters in the output. However, using a heap for retrieving the closest pair of clusters reduces the time complexity of the algorithm to $O(k \cdot n \log n)$.

4. **The Balls algorithm**: The Balls algorithm has an extra input α, which serves

as a threshold for average distance when a cluster and a node are considered for composition. When set to $\frac{1}{4}$, the algorithm provides a clustering of at most 3 times the cost of the optimal. This was proven by Gionis et al [52] in 2007, who initially proposed 3 of the mentioned algorithms (Agglomerative, Furthest, and Balls). The number $\frac{1}{4}$ is not, however, proven to be the optimal value for α. Different values of α are experimented with to find a good approximation of the optimal value for each graph.

The methods are then tested in two real-world scenarios. First, the algorithms are used on a Protein-Protein Interaction network and its corresponding ground-truth data, which resulted in discovering the correct number of clusters and identifying most of the established protein relationships. The results from the experiments indicate that the techniques not only produce meaningful clustering but also discover the correct number of clusters. It is shown that pKwikCluster works better than the other proposed algorithms, as well as the contenders. Also, the practicality of the algorithms is shown using a large social network consisting of one billion edges. This is done to showcase the scalability of pKwikCluster since the previous work could not handle large graphs such as this network.

Despite the good results in practice and the simplicity of the algorithms, there are certain downsides to this method of experimentation. For instance, the experiments are done to showcase the paper's own proposed objective function and the contenders are compared in terms of the defined objective function. However, other measures for the quality of the clustering could be used for comparison. Furthermore, some contenders required the setting of a parameter, which is set to a value that minimizes the expected edit distance.

These downsides are later alleviated by Nilsson [5], who further explains the main shortcomings of the algorithms:

- All algorithms produce a lot of clusters, which makes them impractical when

coarse-grained results are desired, i.e. in visualization.

- Algorithms require the input graph to be complete, which is not realistic in most real-world scenarios.
- The pKwikCluster algorithm is non-deterministic, which could be important in specific applications.
- The furthest algorithm is very slow, the results are not up to par with the rest of the algorithms, and its number of iterations is unpredictable. However, it can perform well when the input graphs are very dense.

Nilsson also [5] studies the effectiveness of the algorithms by implementing them and testing them on a set of fMRI scans. The experiments are done on a set of probabilistic graphs derived from brain scans done on people with or without autism, in order to compare the 4 algorithms. The comparison measures include the number of clusters produced and the balance in cluster size, as well as the mean-intra- and mean-inter-cluster connectivity, with respect to the global connectivity, as suggested by Miasnikof et al. [32]. The Jaccard Index is used to get an overview of the similarities and differences between clusters. The "edit distance" metric [49] is also used for comparison. The study also explains why certain measures are not appropriate to use for evaluation, such as Modularity, the Silhouette Index, and Conductance. However, the datasets in this study are not extensively defined.

Similarity-Based Approach

Similarity-based approaches usually aim to use a similarity measure between nodes to come up with a new representation of the input graph.

For instance, in 2017, Hu et al. [38], used two similarity measures between nodes to acquire an embedding of an arbitrary uncertain graph. This embedding is then used for numerous graph mining applications (i.e. clustering, classification, k-NN querying). The embedding is done in two steps: Proximity matrix computation and

Node vector generation. At first, the graph is converted to its corresponding proximity matrix, which stores the proximity information between every pair of nodes. Two node proximity measures are used in this paper: The "Expected Jaccard Similarity", which is the probabilistic variant of the Jaccard Similarity and is a second-order proximity measure, and "Probabilistic Random Walk with Restart" [22], which is a high-order proximity measure. The random walk procedure is defined as follows: Generate a possible world G for u, walk to a neighbor uniformly at random if there exists any neighbors of u in G, otherwise stay at u. This yields a Probabilistic Transition Matrix which requires additional ways to compute efficiently (the paper has introduced a hash-based method for this computation.) The transition matrix is then used in a matrix relation (Probabilistic Random Walk with Restart) which needs to be computed iteratively by utilizing Monte Carlo methods. These proximity measures are computed using scalable algorithms that are tailored to work on uncertain graphs. Then, the proximity matrix is converted to the node-vector representation of the graph. This is done by solving an optimization problem, utilizing techniques such as negative sampling and asynchronous stochastic gradient descent. The solution to this optimization problem is a factorization of the proximity matrix, which is a compressed form of the initial proximity matrix. This solution can then be used as an embedding of the uncertain graph. The whole embedding algorithm is called *URGE*, which stands for "Uncertain Graph Embedding". Finally, existing graph mining algorithms can be used on the embedding. Extensive experiments show that *URGE* is more effective and faster than the current mining algorithms on uncertain graphs, as well as the state-of-the-art embedding solutions.

Later in 2018, Li. et al. [24] proposed the problem of clustering uncertain graphs with node attributes into k clusters. The definition of Attributed Graphs is given in Section 2.1. The node attributes can be represented as a vector per node. The clustering problem in these networks requires novel ideas because of the challenges imposed by high-dimensional attribute vectors. Li. et al. leverage node attributes

in order to reduce edge uncertainty, which in turn helps with the clustering. Then, each possible world of the uncertain attributed graph is converted to a deterministic weighted graph by using a similarity measure between nodes to convert the attributes to weights. The conversion is done with the help of distance metric learning, to learn the parameters of the Mahalanobis distance [53] between nodes. The learning step aims to find the optimal solution to the optimization problem imposed by the distances. Two methods (AUG-I and AUG-U) are then used to derive a new weighted graph:

- **AUG-I** combines all attribute-induced graphs into a new weighted graph,
- **AUG-U** is based on the unified partition over all possible worlds of an uncertain graph.

In both methods, spectral clustering is used as the final step to cluster the final weighted graph. To evaluate the accuracy of the methods, Accuracy (ACC) and Pairwise F-Measure (PWF) are considered. Furthermore, the reliability of the output clusters is computed using the average clustering reliability (ACR) measure. These metrics are then used to compare the algorithms with several contenders, the most important of which is coded-k-means, by [34].

The results show that the contenders fall behind when measured with the aforementioned metrics. However, the proposed methods have several drawbacks. First, solving the optimization step could be challenging. Second, the algorithms require the input network to have a high assortativity, in order to produce meaningful clusters. Third, there is no way to know the required number of samples needed in the simulation step of the algorithm. Lastly, the contenders are designed for either non-attributed uncertain graphs (which makes them ignore the attributes completely), or deterministic graphs (which makes them ignore the probabilities on the edges). This will cause the proposed algorithms to inherently outmatch the contenders.

Connectivity-Based Approach

Connectivity-based approaches typically involve center-based algorithms that are based on the intuition that a good clustering algorithm focuses on good connectivity on the nodes inside a cluster and low connectivity between the clusters themselves.

In 2017, Ceccarello et al. [35] devised two algorithms using the mentioned intuition to partition the nodes of an uncertain graph into k clusters, each featuring a center node. In summary, approximation algorithms are given for each of these problems:

- **MCP problem**: Maximizes the Minimum Connection Probability of a node to its cluster center, which is proved to be NP-hard.
- **ACP problem**: Maximizes the Average Connection Probability of a node to its cluster center, which is conjectured to be NP-hard.

Both proposed algorithms compute a partial k-clustering. They aim to cover a maximal subset of nodes based on a threshold for the minimum connection probability to a cluster's center.

The algorithm responsible for doing this is called **MIN-PARTIAL**. Given a threshold q on the connection probability, **MIN-PARTIAL** returns a partial k-clustering of an uncertain graph G. This clustering covers a maximal subset of nodes, each connected to its cluster center with a probability of at least q. Nodes not meeting this threshold are considered outliers and remain uncovered.

The two algorithms focus on maximizing the average and the minimum connection probability of any node to its cluster's center, in a possible world. To estimate the connection probabilities among different nodes, Monte-Carlo simulations are utilized. The authors propose an approximation algorithm for metric k-center and k-median problems, using [54] as a baseline. This is extended to probabilistic scenarios to derive a full k-clustering, which is a clustering that covers all nodes.

The main idea is to iteratively call **MIN-PARTIAL** and gradually decreasing q, using specific termination conditions in each of the algorithms. Provable guarantees

are also provided on the quality of the approximation. As an extension to their algorithm, the authors also mention that the proposed algorithms can be run by setting a limit on the length of the paths that contribute to the connection probability between two nodes. This setting is useful in scenarios where the similarity between two nodes diminishes sharply with their topological distance regardless of their connection probability. They emphasize that all the algorithms can be adapted to incorporate limited path lengths d in the estimation of connection probabilities, transforming them into d-hop algorithms.

Finally, the authors compare the algorithm with contenders such as MCL [48] and GMM [55], by using several metrics such as p_{avg}, p_{min}, and Innter and outer AVPR, which are defined in Section 2.3. The results of the experiments show that the clustering that is computed by MCP and ACP feature higher inner-AVPR than GMM and MCL, and a lower outer-AVPR, both of which are desirable features of a good clustering algorithm.

The methods offered in this paper fall short in several areas. In terms of the algorithms, it is not clear which of the metrics (average or min) should be used for the best result in a problem. Also, the maximization problems can have more than one solution, and it is not clear what needs to be done in this case. Furthermore, the contenders that are used in the experiments are primarily designed for weighted deterministic graphs, and they are used on the uncertain graphs by changing the probabilities to weights. This method can be expected to fail, as the contenders are driven more by the topology of the graphs rather than by the connection probabilities. Lastly, the researchers fine-tune the d-hop algorithms by identifying the best d, which makes the results closer to the ground truth.

Later in 2019, Han et al. [56] improved the two algorithms that were initially introduced by Ceccarello et al. [35] for the problems of k-center and k-median. According to the authors, the algorithms introduced by [35] provide rather weak approximation guarantees, regardless of whether the connection probability between any two nodes

is known. Moreover, they state that when good approximation assurance is needed, significant computation overheads are required. Furthermore, the initial k-center algorithm requires knowledge of a lower bound of the optimal value to achieve the claimed asymptotic guarantees; however, the paper does not provide a method for deriving such a lower bound.

Han et al. [56] alleviated these problems by significantly improving approximation and efficiency guarantees, and proving several theorems for the hardness and inapproximability of the problems of k-center and k-median on uncertain graphs. Note that [35] conjectured that the k-median problem is NP-hard when there is a connectivity oracle, but this paper has proved that this conjecture is true. The authors have also utilized the monotonicity and sub-modularity of the objective function of the k-median problem, to devise a greedy algorithm for the problem that leverages graph sampling to achieve better approximations with high probability. Furthermore, a new greedy algorithm is devised for the uncertain k-center problem, which offers better approximations with high probability, and does not require prior knowledge of the optimal value, whereas the initial algorithm requires that a tight lower bound of the value is given.

These algorithms are further improved in practical efficiency by utilizing additional advanced optimization techniques Finally, the authors have defined new measures to quantify the goodness of their clustering in order to compare the algorithms to other similar algorithms in the literature. In the experiments, in order to increase time efficiency, a data structure is used to store the nodes and the generated random samples.

The experimental results demonstrate that the new algorithms outperform the previous ones proposed by [35] on both the processing time and the quality of the clustering results. However, it is worth mentioning that the comparison is only done for [35].

2.5.2 Community Search in Uncertain Networks

Unlike Community Mining, which focuses on partitioning the entire graph, Community Search is more targeted and looks for a specific community that includes the user's query node(s). Community Search is sometimes referred as Local Community Detection [57].

Halim and Khattak [58] proposed a density-based approach, called DBCLPG (Density-Based Clustering of Large Probabilistic Graphs), that starts with a single query node, well-chosen, as a cluster. It then greedily adds nodes from the shell set (also referred to as the cluster neighbors) to the current community, based on a condition to only add "reliable" nodes. The shell set comprises nodes that, while not part of the community, are linked to at least one node within the community, and the reliability of a node is determined by a threshold edge weight (T_w). For each candidate node v, the number of common neighbors between v and the cluster are computed and added to the sum of edges' probabilities (which is the expected degree) between the cluster and v. This value is called the CP (cluster periphery) of the node. If CP is higher than T_w, the node is added to the cluster. The process is repeated until no node in the shell set satisfies the condition. The cluster is then removed from the graph and the procedure is repeated for the remaining graph. This approach is effective for identifying dense communities in uncertain networks, but it may not perform well on networks with complex structures, due to the assumption of high local density for cluster formation. Moreover, the method differs from a typical community search since there is no query node per-se but the ego-centric process starts from a well-chosen node with the aim of partitioning the whole graph. In addition, to carefully choose an optimal node to start with each iteration, the approach needs the full adjacency matrix making it not really solely local.

Qiu et al. [59] propose detailed definitions of density-reachability and structural similarity in probabilistic graphs. The proposed similarity measure can capture the

similarity between two vertices over all the possible instances of the probabilistic graph but is challenging to compute. This challenge is alleviated by using a polynomial-time dynamic programming (DP) algorithm to compute the similarity. The structural clustering problem on probabilistic graphs is then formulated based on this similarity, and the *SCAN* algorithm (which is originally used for deterministic graphs) is adapted to obtain the clustering results. The *SCAN* algorithm is somewhat similar to *SIWO*, in the sense that it starts with a node and checks the neighborhood of that node for potential nodes to add to the cluster. Below we review a few of the important definitions that are used in this work:

- The Structural Neighborhood of a node u (denoted by $N[u]$) in a deterministic graph is defined as the set of neighbors of that node and the node itself.

- The Structural Similarity between two nodes u, v (denoted by $\sigma(u, v)$) is the Jaccard similarity [39] between the set of structural neighborhoods of the two nodes. $\sigma(e)$ is used to denote the structural similarity between u and v in the edge $e = (u, v)$, which can be calculated as the number of common vertices in $N[u]$ and $N[v]$, normalized by $|N[u] \cup N[v]|$:

$$\sigma(u, v) = \frac{|N[u] \cap N[v]|}{|N[u] \cup N[v]|}.$$

  This notion is then used to see if two nodes are similar "enough" (which is called (ϵ-Structural Similarity), by using a similarity threshold ϵ and comparing the structural similarity of the two nodes to ϵ. In probabilistic graphs, however, the notion of ϵ-Structural Similarity will be probabilistic. Determining the ϵ-structurally similarity between two adjacent nodes is done by to identifying the possible worlds in which they are similar and taking a weighted sum, using the probabilities of those possible worlds as weights.

- The (ϵ, η)-reliable neighborhood of u is defined as the subset of vertices like v

in $N[u]$ such that $Pr[(u, v), \epsilon] \geq \eta$. The threshold η is given as input, which is used to filter out the edges that have a probability of ϵ-structural similarity higher than η. In other words, u is called reliable structural similar to v if $Pr(e, \epsilon) \geq \eta$. The definition of (ϵ, η)-reliable neighborhood is in fact used to defined the "good" neighborhood around a node u.

Intuitively, the number of reliable similar neighbors is directly related to the importance of that node in the clustering procedure. Thus, if a node has a sufficient number of reliable similar neighbors, it is called a reliable core node. This is determined by comparing the number of reliable neighbors to another parameter μ. Further details of the algorithm is beyond the scope of this thesis due to complexity. Although the algorithm is shown to have comparable results to the contenders, it is highly parametric which will add layers of unnecessary complexity due to the inevitable hyper-parameter tuning.

In the work by Zhang and Zaïane [47], the authors propose an algorithm called $UR + K$, to address the problem of community detection in uncertain graphs. The algorithm starts with a single node and expands the community by adding neighboring nodes that are considered to be part of the community, using the metrics UR and K. These neighboring nodes that have the potential to be inside the community are referred to as candidate nodes. K measures the closeness of the relationship between a candidate node and an existing community. A high K value indicates that the candidate node is closely related to the community and is more likely to be part of it. K is used in the first few steps of the local community detection algorithm to help choose which neighboring node should be added to the community. UR, on the other hand, stands for the uncertain version of Modularity R [44], is proposed for evaluating local communities in uncertain networks. UR is calculated based on the expected number of edges within the community and the expected number of edges connecting the community to the rest of the network. A high UR value indicates a

sharp boundary between the community and the rest of the network, meaning that the community is well-defined. The algorithm starts by placing the start node in the community and its neighbors in the shell set. At each step, candidate nodes in the shell set are sorted based on their metric (K or UR) values, and the first node that can increase the community's metric is added to the community. This process continues until there are no remaining nodes in the shell node set that can increase the community's metric, and the resulting set of nodes is considered to be a local community. The hyper-parameter λ determines for how many steps K is considered as the main metric before switching to UR. It has been shown experimentally that $UR+K$ outperforms other local community detection algorithms (including ULouvain, the weighted variant of Louvain [60] considering edge probabilities as weights) on real-world and synthetic networks.

DBCLPG [58] and $UR+K$ [47] are among the algorithms used to evaluate the performance of our proposed community search algorithm for uncertain graphs, USIWO, as detailed in Section 4.3.

2.5.3 Community Detection and Search in Deterministic Networks

While our literature review primarily focuses on uncertain graphs, it is essential to acknowledge the extensive body of research dedicated to community detection and search algorithms in deterministic networks. This field has seen significant development and numerous contributions, as evidenced by the comprehensive literature survey conducted by Baltsou et al. [57].

Blondel et al. [60] introduced the Louvain method for community detection, which operates through two primary phases to optimize Modularity Q [43], beginning with the individual assignment of nodes to communities, then relocating them to maximize modularity gains. It is shown that Louvain is relatively fast in finding communities in a network.

However, Louvain had a number of shortcomings, such as the potential for disconnected communities and the resolution limit issue. Later, Traag et al. [61] proposed the Leiden algorithm to address those shortcomings. This algorithm introduces a novel quality function and an efficient node movement strategy termed 'smart local move', significantly reducing unnecessary re-calculations. Additionally, it uses a refinement phase to enhance connectivity within communities and address the resolution limit problem by allowing more flexible community formation. However, the Leiden algorithm also had a number of limitations, such as introducing an additional parameter and potential non-convergence due to the randomness in its refinement phase. Furthermore, both the Louvain and Leiden algorithms are not designed to handle very large graphs.

According to the survey conducted by Baltsou et al. [57], Hamann et al.'s Triangle-Based Community Expansion (TCE) method [62] is the best community search method focusing on node selection, making it a strong benchmark for comparison. TCE is fundamentally based on the Local Tightness Expansion (LTE) algorithm [63], as both exploit the fact that some edges are more embedded in their neighborhood and have more common neighbors than others. LTE uses an edge similarity score based on triangles for deciding which node to add next and for determining the quality of the community. TCE, on the other hand, also uses an edge score based on triangles, but employs conductance for the quality function of the community. Baltsou et al. [57] highlight that experimental evaluations show that TCE exhibits solid performance and often finds the correct community, building on the already excellent performance of the computationally more expensive LTE on most tested graphs. However, a significant drawback of both LTE and TCE is that they use NetworKit[1] for their implementation, requiring the entire graph to be loaded into the main memory, making these algorithms infeasible for handling large networks.

In small networks, the effectiveness of Louvain, Leiden, TCE, and LTE are directly

[1]https://networkit.github.io/

compared against SIWO and other contenders. For detailed information and results of these comparative experiments, refer to [1].

The Local Closest Truss Community (LCTC) method [64], is a community search algorithm shown to handle large graphs. This method addresses the challenge of identifying a Closest Truss Community (CTC) within a network that contains a set of given query nodes. The LCTC method has shown to produce meaningful communities when evaluated on six large real-world networks, making it a suitable benchmark algorithm for assessing SIWO's effectiveness in managing large graphs. The experiments focusing on the scalability of SIWO, along with its comparative analysis with LCTC, are detailed in Section 3.2.1.

2.6 SIWO

The SIWO (Strong Inside, Weak Outside) method was initially proposed as a global community detection technique in [9]. Since its introduction, the algorithm has undergone enhancements, evolving into a local search method with capabilities for global community detection, on both deterministic weighted and unweighted graphs [1]. SIWO aims to optimize a quality function. However, unlike the state-of-the-art methods, its quality function is not an extended version of modularity, which is known for its resolution and field of view limits [9, 40]. We now explore several different notions and definitions that serve as a foundation of SIWO.

2.6.1 Preliminaries

We begin by explaining the notion of "Edge Strength" or "Link Strength" that is introduced in [9]. In the paper, each edge is assigned a strength value in the range of $(-1, 1)$, with stronger edges corresponding to larger weights. The strength of the connection between two nodes is calculated based on the number of neighbors they share, which is referred to as their support value. More specifically, the support of an edge between two nodes is the number of triangles, or size-three cliques, that include

both nodes:

Definition 9 ***(Support)*** *Given a graph $G = (V, E)$, the support of the edge $e_{u,v}$ is the number of mutual neighbors of u and v or the number of triangles that $e_{u,v}$ belongs to, and it is defined as follows:*

$$sup(u, v) = |w \in V, e_{u,w}, e_{v,w} \in E|$$

The strength of the link between two nodes is then calculated based on their support value and the maximum support value of any link involving either node:

Definition 10 ***(Link's strength)*** *Given a graph G=(V, E) and a vertex $u \in V$, the strength $s_{u,v}$ of the link connecting u to a node $v \in V_N(u)$, where $V_N(u)$ is the set of neighbors of u, is defined as follows:*

$$s_{u,v} = sup(u, v)(\frac{1}{sup_{u,max}} + \frac{1}{sup_{v,max}}) - 1$$

Where $sup_{u,max}$ is the maximum support of u and any node in $V_N(u)$: $sup_{u,max} = \max_{w \in V_N(u)}\{sup(u, w)\}$. If $sup_{u,max} = 0$ or $sup_{v,max} = 0$, then $sup(u, v) = 0$ and we assume that part of the equation is zero. We can see that the strength of the connection between a pair of linked nodes increases with the number of mutual neighbors they share.

The concept of Link Strength, which leads to the *SIWO* objective function, forms the foundation of the work presented in [9]. This approach enables community discovery in a network through a greedy optimization process, which iteratively performs two main phases until a local maximum of the SIWO measure is reached. In our version of the SIWO algorithm, these values are calculated locally to find the best candidate nodes to expand the community at each step. Still, this best candidate is added to the community only if it increases the total strength of the edges inside the community [1].

These concepts are among the several foundational concepts that are used in the local variant of the SIWO algorithm. Another concept is the shell set. The shell of a

community C, denoted by $shell(C)$, is the set of nodes that are not in the community but are connected to at least one node in the community.

Formally, the Shell Set is defined as:

Definition 11 (Shell set) *Given a deterministic graph $G = (V, E)$, the shell of the community C is defined by:*

$$shell(C) = \{v \in V, v \notin C \ s.t. \ \exists u \in C, (u, v) \in E\}$$

Another important concept is the notion of Peripheral Nodes. In the context of community detection, these nodes often represent outliers or unique entities that are only loosely connected to a community. Since our method involves counting triangles, peripheral nodes that are not a part of any triangle should be considered at the last stage of our algorithm for potential addition to the found community.

Definition 12 ***(Peripheral Nodes)*** *In a deterministic or uncertain graph, peripheral nodes are nodes that have only one neighbor. These nodes are connected to the rest of the graph through a single edge.*

2.6.2 Algorithm Overview

In *SIWO*, we follow a greedy approach by starting with a query node (or set of nodes), calculating the support and strength values (with Definition 10) for edges between the community (which initially consists of the query node(s)) and nodes in the shell set (which initially consists of the neighbors of the query node(s)), adding the best candidate node from the shell set to the community, and repeating until an stopping condition is met. In the case that the starting query node is a peripheral node, we will consider its sole neighbor as the input to the algorithm instead. The pseudo-code of this algorithm is given in Algorithm 1, and a comprehensive explanation, along with additional information, are given in Chapter 4.

Algorithm 1 *SIWO*: A local community search algorithm

Input: Deterministic Graph $G = (V, E)$, query node(s) $\{q\}$
Output: The community C of the query node(s) $\{q\}$

$C = \{q\}$, $S = V_N(q)$
while $S \neq \emptyset$ **do**
    Calculate and store $s(C, v) = \sum_{u \in C} s_{u,v}$ for all $v \in S$
    Find the node $u \in S$ which maximizes $s(C, u)$
    if $s(C, u) > 0$ **then**
        $C = C \cup \{u\}$
        $S = S \cup (V_N(u) - C) - \{u\}$
    else
        break
    end if
end while
$C = C \cup P$ where P is the set of peripheral nodes that are connected to a node in C
return C

SIWO has been shown to perform effectively for the task of community search. However, in its original form, it only handled moderately-sized deterministic networks. This thesis contributes to adapting SIWO in two major ways:

1. We introduce optimizations that allow SIWO to run on large networks regardless of their file size. This is covered in Chapter 3.

2. We extend the definitions and methodology in SIWO in such a way that it can handle uncertain networks, which we discuss in Chapter 4.

Chapter 3

Enhancing SIWO

In this chapter, we summarize our main contributions to enhancing the SIWO algorithm, with respect to handling very large graphs and its respective experiments. Here, the term *very large graphs* refer to networks characterized by their large scale, typically comprising nodes in the magnitude of millions and edges reaching into the billions. The subsequent sections will explore the methods and experiments that demonstrate how SIWO has been adapted to effectively manage and perform community search on these types of networks.

3.1 Large Graphs and Optimizations

Existing community mining and community search algorithms face challenges when dealing with large graphs, even those designed for local execution. A primary challenge lies in the typical input format of these graphs, as many algorithms either depend on libraries that are designed to load the entire graph into main memory, or they fundamentally lack the capability to selectively read the necessary parts from the graph file. The standard representation of graphs as input files typically consisting of lines that each define an edge with two node IDs (and an associated probability for probabilistic graphs), proves to be inefficient for performing local search on very large graphs. In this format, edge information is scattered throughout the file, without any specific order or structure connecting a node to all its neighbors. This

means that to access all information related to a single node, the algorithm may have to scan through the entire file. This approach becomes increasingly impractical as the graph size grows, as it involves processing large amounts of data just to retrieve information about one node, leading to significant memory and time overheads and processing bottlenecks. Such constraints make this format less suitable for performing local search algorithms on large-scale networks, where the access to a node's complete edge information needs to be done quickly and efficiently.

To mitigate these issues, we introduce unique data structures and optimization techniques, starting with transforming the traditional edge list format of the input graph into an adjacency list format. An adjacency list format facilitates further access to graph entries by storing all connections of each node into a single line in the file, significantly reducing the computational overhead. This transformation is done with the help of a Map-Reduce procedure, and is the first step in addressing the issues of memory inefficiency and slow data retrieval commonly encountered in analyzing large-scale networks.

3.1.1 Map-Reduce Procedure for Graph Conversion

The first step towards optimization is the pre-processing of the input graph. Given the substantial size of these networks, with possibly millions of nodes and billions of edges, it is important to adopt a method for the conversion of the graph that can handle such scale efficiently. The Map-Reduce approach is particularly suited for this task, as it is traditionally used for parallel processing of large datasets.

For **deterministic graphs**, each line in the original format represents an edge in the graph, comprising two entries which are the node IDs. The goal of the Map-Reduce procedure is to convert this format into an adjacency list format, in which the neighborhood information of node i is stored in line i, considering nodes are indexed as positive integers. This conversion ensures that accessing the neighborhood information for any node becomes almost instantaneous when needed.

In details, the original representation of a deterministic graph can be shown as:

Listing 3.1: Original input file for deterministic graphs

```
u_1  v_1
u_2  v_2
.
.
u_n  v_n
```

In this format, each line represents an edge in the graph, with no specific order. The reformatted structure looks like:

Listing 3.2: Converted input file for deterministic graphs

```
1  v_11  v_12  v_13  ...
2  v_21  v_22  v_23  ...
.
.
n  v_n1  v_n2  v_n3  ...
```

In this format:

- Each line corresponds to a node in the graph, represented by a line number (e.g., 1, 2, 3, $\cdots$, n).
- Following the line number, each neighbor of the node corresponding to that line is listed. v_{ij} represents the node ID of the j-th neighbor of node i.

To proceed with the conversion, we use a Map-Reduce procedure. Each line of the input file is initially parsed to extract pairs of nodes representing each edge. The 'Map' phase involves processing these pairs to generate two-way entries, ensuring that each node's connections are symmetrically represented. During the 'Reduce' phase, these node pairs are aggregated by their key, which is the node identifier in this case. The aggregation groups all edges connected to a particular node, creating an adjacency list. This list is then further processed to eliminate any self-loops and duplicate entries. Finally, the adjacency lists for each node are sorted and merged, and the result is saved in a file. Algorithm 2 outlines a summary of the procedure for

deterministic graphs, in the form of a pseudo-code. The procedure is also outlined in Figure 3.1.

Algorithm 2 Deterministic Graph Conversion using Map-Reduce

```
function parseLine(line)
    tokens ← split line into components
    return {(tokens[0], tokens[1]), (tokens[1], tokens[0])}
end function
function GraphConverter(input_file, output_file)
    data ← read input_file into distributed dataset
    data ← apply parseLine on each line in data
    data ← filter out self-loops and duplicates from data
    data ← group data by key (first element of each tuple)
    data ← convert grouped data into list format
    max_node ← find maximum node number in data
    range_data ← create range of numbers from 1 to max_node
    range_data ← parallelize range_data into (node, empty list)
    data ← union of data and range_data
    data ← reduce by key to merge lists
    data ← sort values in each list
    data ← sort data by key (node number)
    data ← convert each entry of data into string
    coalesce data into a single partition
    save data to output_file
end function
```

For **probabilistic graphs**, the reformatted structure and the conversion process is slightly more complex because of the addition of the probabilities, but the goal remains the same: to structure the data in a way that facilitates faster access and processing. Each line in the original format represents an edge in the graph, with the first two entries being node IDs and the third entry being the probability between those two nodes. The Map-Reduce procedure is again employed to convert this format into an adjacency list format, where the neighborhood information of node i is stored in line i.

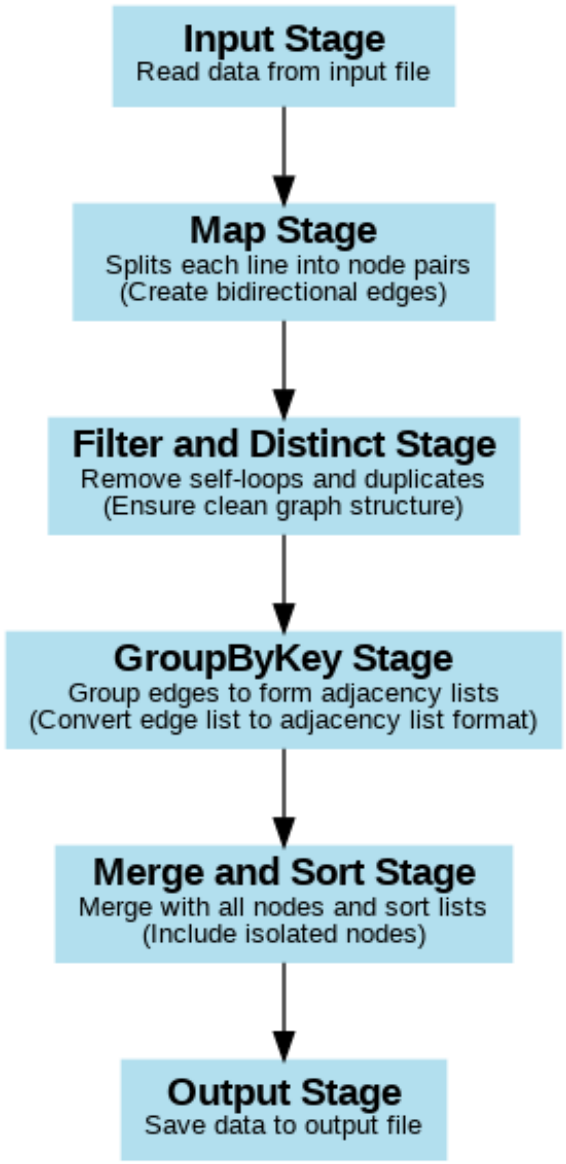


Figure 3.1: Outline of the Map-Reduce procedure used for the conversion of the input file

In details, the original representation of a probabilistic graph can be shown as:

Listing 3.3: Original input file for probabilistic graphs

```
u_1  v_1  p_1
u_2  v_2  p_2
.
.
u_n  v_n  p_n
```

Each line represents an edge in the graph, with the first two entries being node IDs

and the third entry being the probability between those two nodes. The reformatted structure looks like:

Listing 3.4: Converted input file for probabilistic graphs

```
1: {v_11:p_11, v_12:p_12, v_13:p_13, ...}
2: {v_21:p_21, v_22:p_22, v_23:p_23, ...}
3: {v_31:p_31, v_32:p_32, v_33:p_33, ...}
...
n: {v_n1:p_n1, v_n2:p_n2, v_n3:p_n3, ...}
```

In this format:

- Each line corresponds to a node in the graph, represented by a line number (e.g., 1, 2, 3, $\cdots$, n).
- Following the line number and a colon, the neighborhood information of the node corresponding to that line is written in a set of curly braces {}.
- Within the braces of line number i, each neighbor of the node is listed along with the probability of the edge connecting them. v_{ij} represents the node ID of the j-th neighbor of node i, and p_{ij} indicates the probability of the edge between node i and its j-th neighbor.

The Map-Reduce procedure for probabilistic graphs also involves parsing each line of the input file to create pairs of nodes. However, unlike deterministic graphs, we make tuples that include the pair of nodes as well as the probability of the edge between them. The 'Map' phase involves creating a two-way entry for each edge along with its probability. In the 'Reduce' phase, these entries are grouped by node, creating an adjacency list where each node is associated with a list of neighbors and the respective probabilities of the connections.

It is important to note that the conversion process is not limited to enhancing the SIWO algorithm alone, as it can also be utilized by other local community mining or search algorithms. Also note that while this conversion ensures that the input graph representation is in a suitable format for subsequent operations, managing

memory usage while dealing with very large files (which have the size of around tens of Gigabytes) remains a challenge. Even with an optimized graph format, the size of these networks can result in substantial memory consumption if the input file is not handled correctly. To address this, we introduce FastFile, which is a class we used in the enhanced implementation of SIWO.

3.1.2 Memory Optimization: FastFile

Large networks can consume significant memory, making it imperative to design a local approach to read the file and calculate the strengths of only the edges **that are needed**. To achieve this, the FastFile class is proposed, which is specifically designed for efficient reading and interaction with graph data. The primary goal of FastFile is to selectively read and store necessary information from the input file. This is done by utilizing a data structure that consists of a list of line break locations up until the maximum node number that was requested from the file. This approach ensures that only the required part of the graph is stored in memory, thus conserving memory usage. FastFile's approach can be compared to the concept of memory-mapping in operating systems, which maps a section of a file to a part of the virtual address space, reducing the need for repetitive system calls and data copying [65]. For more information regarding managing large files and various file system aspects in operating systems, a comprehensive overview is provided in [66].

Note that the utility of the FastFile class extends beyond its application in the SIWO algorithm, as it can be used as a valuable tool for any implementation dealing with very large input files by providing a way to selectively process only relevant segments of large files.

We now explain how we deal with maintaining time efficiency while processing graph data. This leads us to the introduction of the FastGraph class.

3.1.3 Time Optimization: FastGraph

The FastGraph class represents the graph structure optimized for the SIWO algorithm. It uses dictionaries to store neighbors for each node, support values for pairs of nodes, and strength values for pairs of nodes. By employing these dictionaries, SIWO can quickly check if the required information for a node or edge already exists in the memory. In cases where this information is not pre-stored, the necessary calculations are performed and the dictionaries are updated accordingly.

FastGraph also collaborates with FastFile for efficient graph file access. By using the line break locations stored in FastFile, FastGraph can quickly reach the required line in the file, storing any unseen line break locations in the process. This approach eliminates the need to repeatedly read and process the entire network file, which in turn reduces unnecessary I/O operations and speeds up the algorithm's execution time. By utilizing these two classes, the SIWO algorithm is made truly local by only accessing the lines and computing the strengths that are needed as the algorithm progresses. This is a significant improvement compared to computing strengths for all nodes and edges.

3.1.4 Handling Timeout

One key feature of SIWO is that it can return intermediate results before finishing a search. In other words, SIWO can output the discovered community *up to the allotted time*, which is basically a subset of the targeted community. This is particularly useful for larger graphs where a complete search might be time-consuming. We implemented this feature by utilizing Python's multiprocessing library. By initiating the SIWO algorithm as a separate process, we can impose a timeout constraint, ensuring the algorithm halts exactly after the specified duration. This method was a significant improvement compared to having the algorithm check the timeout in certain breakpoints, which could result in halting too late due to specific computations taking a long time. We introduced a number of shared variables, such as the list of nodes in

the current community, the number of visited nodes, and the current strength of the community. These variables are shared between the separate SIWO process and the main program's process, allowing for real-time updates on their values. If the algorithm exceeds the allotted time, it is terminated, and the community discovered up to that point is returned. This approach ensures that even in scenarios with strict time constraints, SIWO can provide the partial community structure around the query node.

3.2 Running SIWO With a Limited Time Budget and Limited Memory

To compute the quality of the intermediate results by comparing it to the corresponding ground truth using precision, recall, and F_1 Score, we conducted an experiment on a large synthetic network generated with LFR benchmark [67]. The network is generated by setting the number of nodes to 1 million, μ to 0.3, the power law exponent for the degree distribution (τ_1) and and the power law exponent for the community size distribution (τ_2) to 2 and 1 respectively, average degree to 100, and maximum degree to 300. In LFR networks, the mixing parameter μ is the fraction of inter-community edges incident to each node, and controls the fraction of edges that are between communities.

We took each node of a community of 183 nodes as a query node and ran the algorithm several times for different amounts of timeout. Then, for each value of timeout, we took the average of the metrics calculated on each of the output communities. To show the magnitude of the expansion, we also counted the number of nodes visited by the algorithm, as well as the strength and size of the community found so far, per each combination of query node (inside the community) and timeout. The results for this experiment are shown in Table 3.1.

The results show that recall and F_1 Score have a direct relation with the input timeout, which is expected. This means that SIWO is generating meaningful results,

Table 3.1: Quality of SIWO partial search results for given timeout (seconds) in terms of found community size $|C|$, number of nodes visited by algorithm, sum of edge strengths in community $S(C)$, precision P, recall R, and F_1 score ($\pm$ the standard deviation for F_1). Results averaged over 183 runs, using each of 183 nodes as query node in a given ground truth community.

Timeout	|C|	Visited	S(C)	P	R	F_1
10	5	$35,810$	7	1.0	0.027	0.053 ± 0.013
30	29	$209,955$	282	1.0	0.160	0.277 ± 0.011
50	72	$374,825$	1,181	1.0	0.398	0.569 ± 0.023
70	168	$500,589$	$2,154$	1.0	0.922	0.955 ± 0.067
90	182	$512,803$	$2,226$	1.0	0.999	0.999 ± 0.003
100	183	$512,880$	$2,227$	1.0	1.0	1.0 ± 0.0
110	183	$512,880$	$2,227$	1.0	1.0	1.0 ± 0.0

even if there is a limited time. Also, as we give the algorithm more time, it brings nodes that increase the community strength inside the community, until the stopping condition is met (at which point the algorithm will stop even if we give it more time, as illustrated in the last row of the table). To the best of our knowledge, there are no claims regarding the possibility of yielding a result before completion by any of the contender algorithms, and they can not be extended to do so.

Based on the standard deviations, we can also see that the results do not differ greatly by selecting different query nodes inside the same community. In fact, SIWO seems to provide nearly the same results for all query nodes in a community, if enough time is given to the algorithm. This shows that SIWO does not suffer from the issue of the sensitivity of the output community to the initial query node. In other words, the expansion using the SIWO strength function is less likely to cross community boundaries.

Scalability in Large Synthetic Networks

Following the aforementioned improvements, SIWO was able to handle extremely large input graphs regardless of their size. To showcase the scalability, we conducted experiments to observe the performance of SIWO on an even larger synthetic LFR [67] graph consisting of 2 million nodes. The graph was generated with parameters $\tau_1 = 3$, $\tau_2 = 1.5$, $\mu = 0.3$, an average degree of 10, and a maximum degree of 30. This particular choice of parameters, resulted in a network with 154047 communities that is not overly dense, allowing our algorithm to not spend excessive time per node.

We selected 1000 random nodes, ensuring that no two nodes belonged to the same community. The results were again promising, with an average precision of 0.947, an average recall of 0.904, and an average F1 score of 0.920, with all three metrics having an standard deviation of around 0.214. The average time spent per node was 1.768 seconds, with an standard deviation of 0.095. The low average time compared to the experiment on the LFR network with 1 million nodes highlights the truly local nature of our method, demonstrating its capability to efficiently navigate large networks without being hindered by their size. This also indicates that SIWO's time complexity is indeed related to the community size of the given query node and the magnitude of the expansion (determined by the average degree of the nodes in that community), rather than the size of the whole network. This is because SIWO's steps are focused on the local neighborhood around the query node. In the initial stages, the algorithm assesses the strength of connections within the immediate vicinity of the query node, which is a relatively quick process depending on the local density of connections (or the average degree of the nodes). As SIWO expands the community, it considers the neighbors of newly added nodes, but it does this in a way that does not require revisiting the entire network by only considering the new connections at the time of community expansion, instead of recalculating the strength values.

It is crucial to note that the sheer size of this graph necessitates methods that are

Table 3.2: Characteristics of the large real-world networks.

Network name	Number of nodes	Number of edges	Number of communities	Overlap
Youtube [68]	$1,134,890$	$2,987,624$	$8,385$	Yes
Orkut [68]	$3,072,441$	$117,185,083$	$6,288,363$	Yes
UK-2002[1]	$18,520,486$	$298,113,762$	N/A	N/A
Twitter [71]	$41,652,230$	$1,468,364,884$	N/A	N/A
Friendster [68]	$65,608,366$	$1,806,067,135$	$1,449,666$	Yes

truly local and can handle large files without loading them into main memory. This constraint rendered other contenders infeasible for this experiment, as they require substantial memory resources. Our approach's ability to perform efficiently on such a large-scale graph underscores its practicality and robustness in real-world scenarios where memory constraints are often a significant consideration.

3.2.1 Scalability in Large Real-World Networks

Further experiments have also been done to observe the performance of SIWO on five very large real-world networks. The characteristics of these networks (Youtube, Orkut, UK-2002, Twitter, and Friendster) are given in Table 3.2. It is important to note that the available communities for Youtube, Orkut, and Friendster have been functionally defined [68] which do not constitute a confident ground truth [69, 70]. Thus, these networks are used for assessing scalability rather than accuracy.

We compare SIWO to LCTC [64] as it was shown to perform local search on Orkut. However, while the LCTC search algorithm itself is local, it must first create an index of the entire network unlike SIWO which indexes only as needed while the search progresses. The LCTC indexing process requires a significant amount of time and memory to complete, e.g. around 50 GBs of RAM and 4 hours for Orkut, and produces a large index file.

[1] https://law.di.unimi.it/webdata/uk-2002/

Five query nodes were randomly selected from each graph, and subsequently used as the input query node, with 1200 seconds set as the timeout per node. Table 3.3 reports the size of the discovered community $|C|$, the time in seconds and memory in MB used by each algorithm. As SIWO indexes while searching, the reported time includes both the indexing time and the search time. LCTC starts searching after the indexing has been done, so the times can be broken down. For a fair comparison, we compare the total time of indexing plus searching as this would be the time required to find the community given a specific node and network. Nonetheless, even just comparing solely the search time of LCTC with the time SIWO requires, apart from two nodes on Table 3.3 for the Youtube dataset, SIWO was considerably faster. For SIWO, we have also reported the number of nodes that were visited by the algorithm to show the magnitude of the search. The results confirm that SIWO finds larger communities than LCTC in less time while using much less memory. In fact, the memory requirements of LCTC meant that we were not able to run it on the laptop with 16 GB and had to use a machine with much larger memory. Even with this machine we were not able to complete the indexing process of LCTC on the two largest networks and cannot present results. This highlights how SIWO is a truly local algorithm that can handle very large networks on reasonable commodity hardware.

Table 3.3: Performance of SIWO and LCTC on 5 large networks with 5 random query nodes each in terms of maximum memory required (RAM) and size of the index file needed by LCTC (Index Size) in MB, number of nodes visited for SIWO, size of the found community $|C|$ and time in seconds taken by each algorithm. The wall time for a community search is set to 1200 s.

		SIWO				LCTC [64]			
Dataset	Node ID	RAM	Visited	|C|	Time	RAM	Index Size	|C|	Time (indexing + search)
	3		4,999	4	0.8			4	64.0 (49.6 + 14.4)
	2002		11,599	7	2.0			3	64.3 (49.6 + 14.7)
YouTube	19973	7,410	29,400	7	8.5	557	61	2	65.1 (49.6 + 15.5)
	9659		101,026	20	1200.0			8	63.2 (49.6 + 13.6)
	97648		545,630	4,308	943.7			4	63.1 (49.6 + 13.5)
	14347		29	6	0.1			6	15,100.9 (13,663.5 + 1,437.4)
	31025		6,176	357	15.8			3	15,121.8 (13,663.5 + 1,458.3)
UK-2002	78984	1,878	21,851	928	30.1	18,834	2,299	54	15,109.3 (13,663.5 + 1,445.8)
	44379		157,161	1,608	316.3			14	15,097.2 (13,663.5 + 1,433.7)
	61384		335,346	459	496.7			22	15,069.1 (13,663.5 + 1,405.6)
	98171		1,092,307	1,216	522.4			12	20,131.2 (19,438.1 + 693.1)
	2		1,131,011	1,400	392.1			14	20,154.7 (19,438.1 + 716.6)
Orkut	52002	7,419	1,174,254	1,310	418.9	56,934	6,021	19	20,144.2 (19,438.1 + 706.1)
	86525		1,196,757	1,273	572.5			3	20,139.2 (19,438.1 + 701.1)
	79847		1,201,051	1,334	439.3			109	20,163.8 (19,438.1 + 725.7)
	30634		36	9	17.9				
	90205		47	6	22.9				
Twitter	20648	6,918	745	25	51.38		N/A		
	101		1,684	2	184.7				
	43748		17,186	10	53.32				
	72759		1,262	6	54.2				
	18427		1,623	14	46.2				
Friendster	34455	7,428	179,325	104	1200.0		N/A		
	69947		253,145	473	1200.0				
	84063		321,648	730	1200.0				

Chapter 4

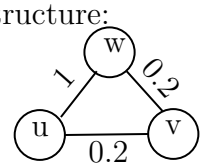
Probabilistic SIWO (USIWO)

In this chapter, we shift our attention to the Probabilistic SIWO, abbreviated as *USIWO* for Uncertain SIWO. Building upon the foundational concepts of SIWO, this extension allows SIWO to run on probabilistic graphs. We begin by detailing the modifications and enhancements made to the original SIWO, including how uncertainty is handled, and how a new stopping threshold was introduced, addressing the gaps identified in the preliminary version of USIWO initially introduced in [72]. We conclude the chapter with an thorough discussion of the results, offering a comparative analysis against other competing algorithms.

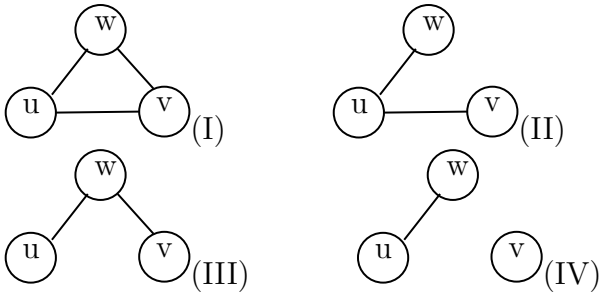
4.1 Preliminaries

USIWO builds upon its deterministic variant SIWO by re-defining the strength formulas. To determine how strong a connection is, we adopt a similar definition of link strengths and support values. The support of an edge between two nodes in a deterministic graph is the number number of distinct triangles, or size-three cliques, containing that edge. This number can be calculated by counting the number of shared neighbors the two nodes have. However, in an uncertain graph, the process of counting these cliques becomes more complex. In the initial introduction of USIWO in [72], the way these values were calculated did not seem to capture an accurate expectation of the triangle count. Rather than counting the shared neighbors, we

must compute their expected count. To illustrate how we can modify the support value formula to account for probabilistic cases, consider the following probabilistic structure:



To count the expected number of triangles between u and v, we need to check all possible worlds. In all possible worlds, there is an edge between u and w, so we only need to consider the other two edges. There are 4 possible worlds corresponding to the existence and non-existence of these two edges:



We can observe that among these four possible worlds, only one of them (I) contains a triangle between the three nodes. With u, v, w there are a maximum of $2^3 = 8$ possible worlds regardless of their edge probability, and only one possible world will have the 3 edges occurring (i.e. a triangle). The probability of that possible world occurring is the product of the probability of the three edges:

$$P = 1 \times 0.2 \times 0.2 = 0.04$$

Given the fact that only one possible world will have the 3 edges occurring (i.e. a triangle), the expected number of distinct triangles $E[N_{u,v}|w]$ between nodes u and v related to w is defined as follows:

$$E[N_{u,v}|w] = (p_{u,w} \times p_{w,v} \times p_{u,v})$$

Where $p_{i,j}$ is shorthand for $p(e_{i,j})$, or the probability assigned to the edge between i and j in the graph. Having defined this notion, we can now define the probabilistic

support. In a probabilistic graph, the support value of an edge, corresponding to the expected number of triangles between the two linked nodes, is defined as follows:

Definition 13 ***(Probabilistic Support)*** *Given an uncertain graph $\mathcal{G} = (V, E, p)$, the support of the edge $e_{u,v}$ is the expected number of mutual neighbors of u and v or the number of triangles that $e_{u,v}$ belongs to, and it is defined as follows:*

$$sup(u, v) = \sum_{w \in V_N(u) \cap V_N(v), w \notin \{u,v\}} E[N_{u,v}|w]$$

Where $V_N(u)$ denotes the neighborhood of the node u defined as the nodes in the graph for which there is a non-zero probability of an edge between them and u. It is important to note that the nodes w contributing to the summation must be part of the shared neighborhood of both nodes u and v. This is because if a node w is not a mutual neighbor of u and v, either $p_{u,w}$ or $p_{v,w}$ would be zero, rendering the entire term zero in the summation. Therefore, only those nodes w that are mutual neighbors of u and v contribute non-zero terms to the summation, reflecting their role in the formation of potential triangles involving u and v.

The strength of the link between two nodes is calculated similar to the deterministic case based on their support value and the maximum support value of any link involving either node. The link strengths are scaled in such a way that they take a value in $[0, 1]$:

Definition 14 ***(Probabilistic Strength)*** *Given an uncertain graph $\mathcal{G} = (V, E, p)$, and a vertex $u \in V$, we define the strength of the link connecting u to a node $v \in V_N(u)$ as follows:*

$$s_{u,v} = \frac{sup(u, v)}{2}(\frac{1}{sup_{u,max}} + \frac{1}{sup_{v,max}})$$

Where $sup_{u,max} = \max_{w \in V_N(u)}\{sup(u, w)\}$ is the maximum support of u and any node in $V_N(u)$.

Table 4.1: Characteristics of the small real-world networks.

Network name	Number of nodes	Number of edges	Number of communities	Overlap
Karate [73]	34	78	2	*No*
Dolphins [74]	62	159	2	*No*
Football [7]	115	613	12	*No*

4.2 Methodology

In this section, we propose the detailed methodology of our proposed USIWO algorithm for community search in uncertain graphs. Building upon the foundational concepts and definitions of support and strength values established in the previous section, we outline the step-by-step process of our approach. This includes the initialization phase, the iterative process of node expansion, and the criteria for stopping the expansion.

USIWO begins by placing the query node in an empty community. It then locally explores the network to identify the most suitable node in the community's neighborhood to expand the community one node at a time. This iterative process continues until the desired community around the given query node is found. The method involves five key steps, as outlined in Algorithm 3:

1. **Update Shell Set**: The shell set, initially empty, is updated after placing the query node u in the community C. All nodes connected to u are added to the shell S. If the query node is a peripheral node, its single neighbor replaces the query node. The shell set is updated in subsequent rounds by removing the node v that joined C in the previous round and adding nodes directly connected to v that are not already in C. The nodes in the shell set are called "candidate nodes", since they represent potential candidates for new nodes that may be added to the community. In the uncertain setting, the shell set is defined similar

to the deterministic case:

Definition 15 (Shell set (Probabilistic)) *Given an uncertain graph* $G = (V, E, p)$*, the shell of the community* C *is defined by:*

$$shell(C) = \{v \in V, v \notin C \ s.t. \ \exists u \in C, p(e_{u,v}) > 0\}$$

2. **Assign Strength Values**: Following Def. 14, the strength value $s_{u,v}$ is computed for each pair of nodes (u, v) where $u \in C$ and $v \in S$. This process is performed locally, meaning it does not require access to the entire network. We can also make sure we only calculate what is needed in each step and store the calculated strength values in a data structure, to avoid re-calculation of values.

3. **Select Best Candidate Node**: After edge strengths are determined, the strength that each potential candidate node v can bring to the current community is computed. This value, denoted by $s(C, v)$, is the sum of the strength values of all edges between the community and v:

$$s(C, v) = \sum_{u \in C} s_{u,v}$$

The node corresponding to the largest $s(C, v)$ is declared the best candidate.

4. **Expand Community**: The community is expanded by adding the selected node, but only if its addition will result in a significant increase in the overall strength of the community. This "significance" is determined by the stopping threshold δ, which is a threshold that can be set depending on how large we want the resulting communities to be. Thus, for the best candidate node v, we check if $s(C, v) > \delta$. In that case, the algorithm returns to the first step to update the shell set. If no new node can be added, (i.e., there is no node v such that the sum of the strength values between v and the nodes inside the community would exceed δ), the algorithm proceeds to the final step.

5. **Reform Community**: The final step of the algorithm is to add any peripheral node that has a neighbor in the current community. Without this step, these nodes do not have the chance to join the community, because the edge that connects them to the rest of the graph cannot be a part of any triangle.

The proposed method of greedy expansion ensures that the final detected community is a connected sub-graph and that a node joins C only if it improves its strength. The process repeats until an optimal community structure is obtained. Furthermore, to make the algorithm truly local, USIWO only loads the necessary edges and calculates their strengths by utilizing the techniques mentioned in Section 3.1. Although this greedy approach provides us with a community "search" algorithm with the objective of finding the best community around a given query node, it can be extended to community mining by repeatedly applying the community search process to random query nodes and removing the found community from the network, defining it as a distinct community (or cluster). This strategy offers a practical approach to community detection in uncertain graphs, particularly when dealing with large graph structures. Algorithm 3 outlines a summary of the proposed USIWO algorithm in the form of a pseudo-code.

The selection of an appropriate stopping condition, combined with the probabilistic support values, can yield optimal results, provided a suitable threshold value is chosen. In Section 4.3, we provide evidence supporting this claim and detail how we determined an appropriate threshold value through a series of experiments.

4.3 Experiments and Results

The aim of the experiments is to assess the performance of USIWO in comparison with other existing methods.

Algorithm 3 *USIWO*: A local community search algorithm

Input: Uncertain Graph $G = (V, E, p)$, query node(s) $\{q\}$, and the stopping threshold δ
Output: The community C of the query node(s) $\{q\}$

$C = \{q\}$, $S = V_N(q)$
while $S \neq \emptyset$ **do**
  Calculate and store $s(C, v) = \sum_{u \in C} s_{u,v}$ for all $v \in S$
  Find the node $u \in S$ which maximizes $s(C, u)$
  if $s(C, u) > \delta$ **then**
    $C = C \cup \{u\}$
    $S = S \cup (V_N(u) - C) - \{u\}$
  else
    break
  end if
end while
$C = C \cup P$ where P is the set of peripheral nodes that are connected to a node in C
return C

4.3.1 Datasets

Due to the scarcity of probabilistic networks with ground truth available online, we employed a method to convert deterministic graphs into uncertain graphs. Our method is a variation of the network generator proposed in [72], which is built upon the network generator introduced in [47]. This generation process was chosen for its ability to create large synthetic uncertain graphs, allowing us to compare the output with the ground truth. In particular, we use this method to convert LFR [67] networks and real-world networks with ground truth to uncertain graphs. The generation process is outlined below:

- The algorithm takes a deterministic network G with the ground-truth communities as input. It also takes two parameters: p_{intra} and p_{inter}. These parameters define the range of possible probability values for intra-community links and inter-community links respectively. Both p_{intra} and p_{inter} are bounded between 0 and 1, where p_{intra} defines the lower bound for intra-community link probabil-

ities and p_{inter} defines the lower bound for inter-community link probabilities. The value for p_{intra} is commonly expected to be larger than or equal to p_{inter}, so that the generated intra-community links would be generally more probable than inter-community links.

- The process of converting deterministic networks into uncertain networks begins by identifying the links as either intra-community links or inter-community links based on the ground-truth communities. For each intra-community link, a probability value is generated using a uniform distribution, ranging between the p_{intra} value and 1. Similarly, for each inter-community link, a probability value is generated using the same uniform distribution, but between the p_{inter} value and 1.

- The generated probability values are then assigned to their respective links, turning the deterministic network into an uncertain network while preserving the community structure from the original deterministic network and does not compromise the preset ground truth.

For the purpose of our experiments, we also introduce the concept of "complexity" to describe different variations of added uncertainty. The complexity of an uncertain graph is determined by the parameters p_{intra} and p_{inter}, as well as the method used to assign probability values to the edges. We define four levels of complexity:

1. **Complexity 1**: Probability values are first generated uniformly at random in range of [0,1]. These values are then sorted from most likely to least likely and assigned to intra-community edges and inter-community edges, respectively. This ensures that intra-community edges are always stronger and the ground truth is not compromised.

2. **Complexity 2**: The lower bound for intra-community link probabilities (p_{intra}) is set to 0.6, while the lower bound for inter-community link probabilities (p_{inter})

is set to 0. This creates a scenario where intra-community links (which are given probabilities in range $[0.6, 1]$) are generally stronger than inter-community links (with probabilities in range $[0, 1]$).

3. **Complexity 3**: Both p_{intra} and p_{inter} are set to 0.6, creating a scenario where both intra-community and inter-community links can be strong.

4. **Complexity 4**: In addition to the settings of Complexity 3, new probabilistic edges are added between communities with a low probability (0.01). This introduces additional uncertainty and can potentially make community detection more challenging.

The complexity levels were designed to progressively increase the difficulty of the community detection task, and create multiple variations of uncertain networks from a given deterministic network. Complexity 1 represents the simplest scenario, where intra-community edges are always stronger than inter-community ones. Complexity 2 introduces some uncertainty by allowing inter-community edges to be potentially as strong as intra-community edges. Complexity 3 further increases the uncertainty by making the strength of intra- and inter-community edges indistinguishable on average. Finally, Complexity 4 represents the most challenging scenario, where new low-probability edges are added between communities, further blurring the community boundaries. This way, we are able to examine the performance of community detection algorithms under different uncertainty scenarios.

Real-world Networks

For the experiments on the real-world graphs, we consider three well-known networks with a known community structure, namely, the Karate [73], Football [7] and Dolphins [74] networks. These networks were also used as potential graphs for experiments in works of [75] and [47]. Each of these deterministic networks are then converted to four uncertain networks using the explained method.

Synthetic Networks

To further evaluate the algorithms on a network with ground truth, we used our conversion method to convert an LFR network [67] with 2500 nodes into a probabilistic network. The parameters for generating the LFR network were set as follows: the power law exponent for the degree distribution τ_1 was set to 3, the power law exponent for the community size distribution τ_2 was set to 1.5. These were selected in such a way that the network would have a realistic degree distribution and community size distribution, respectively. The fraction of inter-community edges incident to each node μ was set to 0.2, which would create a significant yet manageable level of overlap between communities. The average and maximum degree of nodes were set to 10 and 30 respectively, to cause the network to have a moderate density. Additionally, to test scalability, we used the same parameters to generate a much larger LFR network with 1 million nodes and 5,774,105 edges. This time, we chose 1 percent of all edges and sampled their intra-community edge probabilities and inter-community edge probabilities from [0.7, 1.0] and [0.0, 0.3], respectively.

4.3.2 Optimal Threshold Value

Before testing the effectiveness of our method on the converted graphs, we conducted a series of experiments to determine the optimal threshold value for the stopping condition. These experiments were performed in two stages. In the initial stage of our experiments, we explored a range of threshold values, incrementing by 0.1 from 0.1 to 1.5. We applied the algorithm to several query nodes and evaluated the average F_1 measure against the threshold for three real-world graphs: Football, Karate, and Dolphins. For each graph, one query node was randomly selected from each community. These experiments suggested that an effective threshold value consistently fell within the range of 0.8 to 0.9, regardless of the input graph or the specific nodes chosen. To further refine our threshold selection, we conducted an additional set of experiments, adjusting the threshold between 0.8 and 0.9 in increments of 0.01. The

outcomes of these experiments led us to identify an optimal threshold value of 0.82 within this range. This value was then applied as the stopping condition for the USIWO algorithm in all subsequent experiments.

4.3.3 Competitors

When it comes to community search in uncertain graphs, the number of direct competitors is limited. The most notable method in this category is the UR+K method proposed by Zhang and Zaïane [47]. We ran the community search algorithms, namely USIWO and UR+K, for each query node in the uncertain graphs. The resulting communities were then compared with the ground truth communities. We calculated the usual precision, recall, and F_1 measures for the results of each query node. The final reported metrics for these algorithms are the averages of these individual results.

Given the scarcity of direct competitors in the field of community search, we also included in our comparison other algorithms that are primarily used for community mining. This approach allows us to provide a more comprehensive evaluation of our method. We first considered the DBCLPG method proposed by Halim and Khattak [58], which has shown promising results in detecting communities in uncertain graphs. Furthermore, we also included the Leiden algorithm [61] in our comparison. The Leiden algorithm is widely recognized as one of the most effective algorithms for community detection in deterministic graphs, and it has been shown to outperform the Louvain algorithm [60]. We included both the weighted and unweighted versions of the Leiden algorithm in our comparison. Even though the Leiden algorithm is originally designed for deterministic graphs, for the purpose of our experiments, we adapted it to work with uncertain graphs. For the weighted version of the Leiden algorithm, we treated the edge probabilities as weights. For the unweighted version, we ignored the edge probabilities altogether and treated all edges as if they do not have a weight. This approach allowed us to apply the Leiden algorithm to uncertain graphs and include it in our comparison.

For the community mining algorithms, which include the two versions of the Leiden algorithm and DBCLPG, and for each community found by these algorithms, we associated each discovered community with all its contained query nodes. We then proceeded as before, computing the precision, recall, and F_1 measure for each query node, then averaged these metrics to obtain the final results. Note that choosing bad starting nodes, such as those on the boundaries of a community, could potentially lead to poor results. By attributing the communities found via the optimal starting node to all nodes inside that community instead, this starting node issue is mitigated, which gives these methods an advantage over USIWO and UR+K.

4.3.4 Results and Discussion

To evaluate the performance of our algorithm, we ran USIWO on each node of the three real-world networks (Karate, Football, and Dolphins), and on 100 random nodes of the synthetic network. We then calculated and recorded the precision, recall, and F_1 measure, and calculated the average of these measures across the nodes. Figures 4.1 and 4.2 plot the average F_1 scores for each algorithm on each uncertain graph, and their corresponding standard deviations. The average results over existing graphs are shown in Table 4.2.

For the real-world networks, the USIWO algorithm outperforms other algorithms in terms of Recall, and F_1 measure on average and it is second in terms of Precision as shown in Table 4.2. However, in specific cases such as the Football network at complexity level 4, the Weighted Leiden algorithm surpasses USIWO with an F_1 score of approximately 0.77, compared to USIWO's score of 0.65. This can be attributed to the "starting node issue", which causes both versions of the Leiden algorithm to start with good starting nodes and having all nodes of a community contribute to a better performance, whereas the new added links in graph with complexity level 4 may cause USIWO to start venturing on a wrong community when starting with a bad node, leading to poor results for that specific node. This kind of scenario does

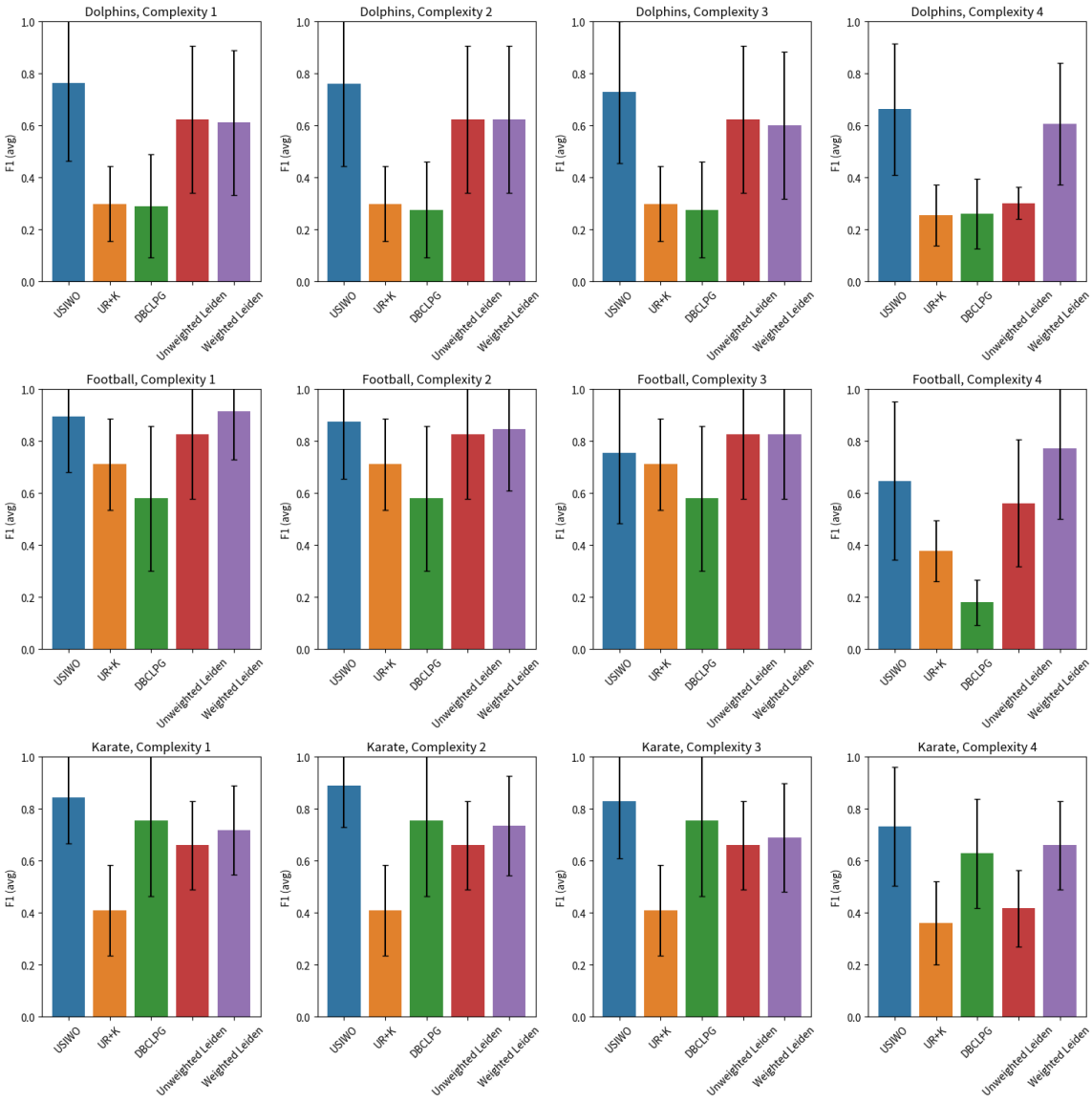


Figure 4.1: Community search: Comparing the F_1 scores calculated over all nodes used as query nodes on real world networks. Each bar represents the average F_1 score for the corresponding algorithm, with the error bars representing the standard deviations.

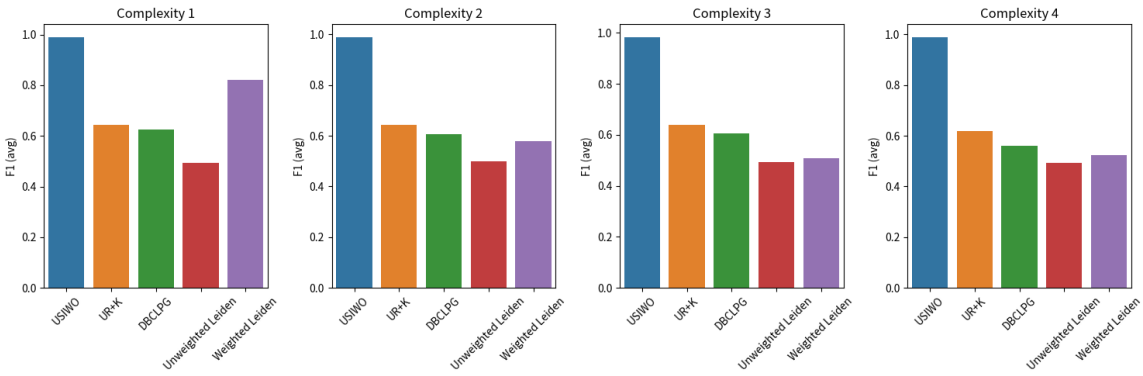


Figure 4.2: Community search: F_1 average scores calculated over 100 nodes used as query nodes on a synthetic network

not happen in community mining algorithms that start with the most optimal node.

Despite this, USIWO shows its strength in consistently achieving high scores across a range of networks and complexity levels. Specifically, USIWO has the highest performance in 9 out of the 12 experiments regarding the real-world networks, and in all of the experiments regarding the synthetic networks.

For the small-scale synthetic network, USIWO demonstrates outstanding performance, achieving an average precision of nearly 1.00, a recall of 0.98, and an F_1 score of 0.99. The low standard deviation for the F_1 score, 0.04, suggests that USIWO's performance is stable and reliable across different query nodes.

The runtimes of the algorithms are also reported as the experiments were conducted on a commodity laptop. The UR+K algorithm and both variations of the Leiden algorithm had an average runtime of approximately 0.03 seconds. The DBCLPG algorithm was the slowest, taking about 1.91 seconds on average. USIWO had a runtime of approximately 1.41 seconds, which, although longer than UR+K and the Leiden algorithms, is still reasonable given its superior performance in terms of precision, recall, and F1 score. However, the runtimes drastically change when the size of the network increases. The real strength of USIWO comes from its scalability.

To test scalability, an experiment is done on the 1M node synthetic network. On this network, USIWO averaged 1.605 seconds per node (for 100 searches) with preci-

sion, recall, and F1 scores equal respectively to 1.000, 0.988, and 0.994, using only 200 MBs of RAM. In comparison, UR+K took 19.03 seconds per node, achieving scores of 0.706, 0.555, and 0.598 in precision, recall, and F1, respectively, while consuming around 5 GBs of RAM. Furthermore, Weighted and Unweighted Leiden, as well as DBCLPG, faced memory errors and could not operate on this network, mainly due to the way they handle the input graph.

Our results experimentally validate the fact that the strength values calculated using Definition 14 effectively capture a node's relative importance relative to its other connections, providing a perspective that is both consistent and reflective of broader connectivity patterns of the involved nodes.

Table 4.2: Performance on real-world and Synthetic networks

Algorithm	Precision (avg)	Recall (avg)	F1 (avg ± std)
Real-World networks			
DBCLPG	0.78	0.46	0.49 ± 0.23
UR+K	0.75	0.42	0.44 ± 0.16
USIWO	0.83	**0.80**	**0.78** ± 0.25
Unweighted Leiden	0.82	0.60	0.63 ± 0.21
Weighted Leiden	**0.92**	0.66	0.72 ± 0.23
Small-scale Synthetic Network			
DBCLPG	0.71	0.56	0.60 ± 0.31
UR+K	0.72	0.60	0.64 ± 0.12
USIWO	**1.00**	0.98	**0.99 ± 0.04**
Unweighted Leiden	0.36	1.00	0.50 ± 0.21
Weighted Leiden	0.48	1.00	0.61 ± 0.20

Chapter 5

Conclusion and Future Work

In this thesis, we explored the concept of community search in large uncertain and deterministic networks, emphasizing the SIWO algorithm and its adaptability to these networks. The primary achievements and contributions gained through this work are summarized below.

5.1 Summary of Contributions

1. **Efficient Handling of Large Graphs:** We developed a methodology to convert large graphs into a format optimized for local community search algorithms. By doing so, we significantly reduced the resources needed for the processing of massive networks by eliminating the need for storing the entire network in main memory. This advancement is detailed in Chapter 3.

2. **Enhanced SIWO Algorithm:** The SIWO algorithm underwent significant enhancements in this work. We utilized data structures and optimization techniques that were specifically chosen for their efficiency in dealing with large datasets. This new implementation of SIWO also makes it capable of operating within specific time constraints and producing accurate partial results. These enhancement are described in Chapter 3.

3. **Adaptation to Uncertain Networks:** A significant part of this work is focused on adapting the SIWO algorithm for uncertain networks. This adaptation

was necessary to address the complexities introduced by the probabilistic nature of these networks. This is discussed in Chapter 4.

4. **Experimental Validation:** We presented several experiments that showcase the scalability and performance of both the enhanced SIWO and the adapted USIWO algorithms. These experiments, covered in Chapters 3 and 4, provided empirical evidence of the algorithms' effectiveness in synthetic and real-world networks.

5.2 Potential Areas for Future Research

In this section, we present various directions for future research on the topics discussed in this thesis. The future research areas outlined in this section hold the potential to significantly advance the field of community search in large deterministic and uncertain networks. As networks grow in complexity and size, the emphasis will be on developing more sophisticated, efficient, and versatile tools that can adapt to different kinds of networks.

5.2.1 Enhancing Scalability and Efficiency

While our approach is local and by definition only explores locally without loading the whole graph, future research could focus on further enhancing the scalability and efficiency of SIWO and USIWO. This might involve exploring more efficient data structures or parallel processing techniques to handle the computational challenges of handling large-scale deterministic and uncertain graphs. By utilizing parallel computing and distributed systems, we can enhance the SIWO and USIWO algorithms to operate efficiently on distributed computing platforms. This would make them more scalable and efficient, and potentially more suitable for handling very large networks.

5.2.2 Expanding Applicability

The applicability of SIWO and USIWO can be explored in other types of graphs, such as weighted uncertain graphs and attributed uncertain graphs, as well as graphs with uncertainties on nodes in addition to edges. This may also include an adaptation of SIWO and USIWO for real-time community detection in dynamic and evolving networks, where nodes and edges change over time. Furthermore, SIWO and USIWO can also be expanded to work in multimodal networks [76], where nodes can represent different types of entities and edges can represent different types of relationships.

5.2.3 Integration with Machine Learning Algorithms

Future research could explore the integration of machine learning algorithms with SIWO and USIWO, to develop methods for automated tuning of the algorithm parameters, such as the stopping thresholds or introducing new thresholds for edge or community strength. Specifically, it can be used to train a more accurate stopping condition that can adjust the stopping threshold value based on the network, determining when the best candidate node does not have "enough" strength. This automated tuning would make the algorithms more applicable in different types of networks, without the need to manually adjust the threshold.

5.2.4 Development of a Comprehensive Software Suite

Developing a comprehensive software suite that includes the enhanced SIWO and USIWO algorithms along with visualization tools and data pre-processing modules could be a significant contribution. Such a suite would make the algorithms more accessible and user-friendly for network analysts and researchers.

5.2.5 Experiments on Robustness

Investigating how SIWO and USIWO perform under varying data conditions can be beneficial to assess their robustness and adaptability. Research could focus on

how these algorithms handle networks with different levels of connectivity density and node degree distribution. Furthermore, even though the current experiments performed with USIWO include various levels of uncertainty, conducting a thorough quantitative analysis of how different levels and types of uncertainty in networks affect the performance of the algorithm could be useful. This research would involve systematically varying the degrees of uncertainty in a wider variety within benchmark networks to examine how these variations affect the accuracy and efficiency of the algorithms.

5.3 Closing Remarks

In summary, this thesis has made substantial contributions to the understanding and practical implementation of community search algorithms in both large deterministic and uncertain networks. However, the field of network analysis is constantly changing, bringing about new challenges and opportunities. Therefore, continuous research in this area is crucial to stay updated with the evolving characteristics of complex networks.

Bibliography

[1] M. Zafarmand, Y. Talebirad, E. Austin, C. Largeron, and O. Zaïane, "Fast local community discovery relying on the strength of links," *Social Network Analysis and Mining*, vol. 13, Sep. 2023.

[2] S. Sakr *et al.*, "The future is big graphs: A community view on graph processing systems," *Commun. ACM*, vol. 64, no. 9, pp. 62–71, Aug. 2021.

[3] X. Huang, L. V. Lakshmanan, and J. Xu, *Community Search over Big Graphs: Models, Algorithms, and Opportunities*. 2017, pp. 1451–1454.

[4] A. Khan, Y. Ye, and L. Chen, *On Uncertain Graphs* (Synthesis Lectures on Data Management). Springer International Publishing, 2022.

[5] A. Nilsson, *Implementing and evaluating clustering methods for large probabilistic graphs*, Available Online, 2021.

[6] C. Zhang and O. Zaïane, "Neighbor-based link prediction with edge uncertainty," in Mar. 2019, pp. 462–474.

[7] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *PNAS*, pp. 7821–7826, 2002.

[8] P. Hintsanen and H. Toivonen, "Finding reliable subgraphs from large probabilistic graphs," *DMKD*, pp. 3–23, 2008.

[9] S. Z. Gharaghooshi, O. R. Zaïane, C. Largeron, M. Zafarmand, and C. Liu, "Addressing the resolution limit and the field of view limit in community mining," in *IDA*, 2020, pp. 210–222.

[10] E. Austin, "Topic modelling via community mining of term co-occurrence networks," Available Online, M.S. thesis, University of Alberta, 2022.

[11] S. Fortunato and M. E. J. Newman, "20 years of network community detection," *Nature Physics*, vol. 18, no. 8, pp. 848–850, Aug. 2022.

[12] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications* (Structural Analysis in the Social Sciences). Cambridge University Press, 1994.

[13] T. A. B. Snijders, "Social network analysis," in *International Encyclopedia of Statistical Science*, M. Lovric, Ed., Springer, 2011, pp. 1356–1358.

[14] S. P. Borgatti, A. Mehra, D. J. Brass, and G. Labianca, "Network analysis in the social sciences," *Science*, vol. 323, no. 5916, pp. 892–895, 2009.

[15] A. Barrat, M. Barthelemy, R. Pastor-Satorras, and A. Vespignani, "The architecture of complex weighted networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, pp. 3747–3752, Apr. 2004.

[16] I. Falih, N. Grozavu, R. Kanawati, and Y. Bennani, "Community detection in attributed network," in *Companion Proceedings of the The Web Conference*, Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018, pp. 1299–1306.

[17] P Ramesh, J JebaEmilyn, S Vasanthi, M Venkatesh, and J. Aldo Stalin, "Detailed investigation: Performance of influence analysis towards big social data," in *Proceedings of the International Conference on Intelligent Computing Systems*, 2017.

[18] D. Cartwright and F. Harary, "Structural balance: A generalization of heider's theory," *Psychological Review*, vol. 63, no. 5, pp. 277–293, 1956.

[19] Y. Ding, "Scientific collaborfation and endorsement: Network analysis of coauthorship and citation networks," *Journal of Informetrics*, vol. 5, no. 1, pp. 187–203, 2011.

[20] R. Jin, L. Liu, and C. C. Aggarwal, "Discovering highly reliable subgraphs in uncertain graphs," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery, 2011, pp. 992–1000.

[21] Y. Gu, C. Gao, G. Cong, and G. Yu, "Effective and efficient clustering methods for correlated probabilistic graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 5, pp. 1117–1130, 2014.

[22] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios, "K-nearest neighbors in uncertain graphs," *Proc. VLDB Endow.*, vol. 3, no. 1–2, 2010.

[23] D. Suciu, "Probabilistic databases," in *Encyclopedia of Database Systems*, 2009.

[24] Y. Li, X. Kong, C. Jia, and J. Li, "On clustering uncertain graphs with node attributes," in *ACML*, 2018.

[25] S. Banerjee, "A survey on mining and analysis of uncertain graphs," *Knowledge and Information Systems*, vol. 64, no. 7, pp. 1653–1689, Jul. 2022.

[26] M. Coscia, F. Giannotti, and D. Pedreschi, "A classification for community discovery methods in complex networks," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 4, no. 5, pp. 512–546, 2011.

[27] P. De Meo, E. Ferrara, G. Fiumara, and A. Provetti, "Mixing local and global information for community detection in large networks," *Journal of Computer and System Sciences*, no. 1, pp. 72–87, 2014.

[28] D. Luo, Y. Bian, Y. Yan, X. Liu, J. Huan, and X. Zhang, "Local community detection in multiple networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 266–274.

[29] M. Takaffoli, R. Rabbany, and O. R. Zaiane, "Incremental local community identification in dynamic social networks," in *IEEE/ACM International Conference on Social Networks Analysis and Mining*, 2013.

[30] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *J. Mach. Learn. Res.*, vol. 11, pp. 2837–2854, Dec. 2010.

[31] M. E. J. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[32] P. Miasnikof, A. Y. Shestopaloff, A. J. Bonner, Y. Lawryshyn, P. M. Pardalos, and E. Estrada, "A density-based statistical analysis of graph clustering algorithm performance," *Journal of Complex Networks*, vol. 8, no. 1, pp. 1–33, 2020.

[33] C. J. Colbourn, *The combinatorics of network reliability* (The International series of monographs on computer science 4). New York: Oxford University Press, 1987.

[34] L. Liu, R. Jin, C. Aggarwal, and Y. Shen, "Reliable clustering on uncertain graphs," in *ICDM*, 2012.

[35] M. Ceccarello, C. Fantozzi, A. Pietracaprina, G. Pucci, and F. Vandin, "Clustering uncertain graphs," *VLDB*, pp. 472–484, 2017.

[36] A. Strehl and J. Ghosh, "Cluster ensembles — a knowledge reuse framework for combining multiple partitions," *J. Mach. Learn. Res.*, vol. 3, pp. 583–617, 2002.

[37] T. Yang, R. Jin, Y. Chi, and S. Zhu, "Combining link and content for community detection: A discriminative approach," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009, pp. 927–936.

[38] J. Hu, R. Cheng, Z. Huang, Y. Fang, and S. Luo, "On embedding uncertain graphs," in *CIKM*, 2017, pp. 157–166.

[39] P. Jaccard, "Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines.," *Bulletin de la Societe Vaudoise des Sciences Naturelles*, vol. 37, pp. 241–72, Jan. 1901.

[40] A. Lancichinetti and S. Fortunato, "Limits of modularity maximization in community detection," *Physical Review E*, vol. 84, no. 6, Dec. 2011.

[41] S. Fortunato and M. Barthélemy, "Resolution limit in community detection," *Proceedings of the National Academy of Sciences*, vol. 104, no. 1, pp. 36–41, 2007.

[42] U. Brandes *et al.*, "On modularity clustering," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 20, pp. 172–188, Mar. 2008.

[43] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, pp. 26–113, Feb. 2004.

[44] A. Clauset, "Finding local community structure in networks," *Physical Review E*, vol. 72, no. 2, pp. 26–132, Aug. 2005.

[45] F. Luo, J. Wang, and E. Promislow, "Exploring local community structures in large networks," vol. 6, Jan. 2008, pp. 387–400.

[46] J. Chen, O. R. Zaïane, and R. Goebel, "Detecting communities in social networks using local information," pp. 197–214, 2010.

[47] C. Zhang and O. R. Zaïane, "Detecting local communities in networks with edge uncertainty," in *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2018, pp. 9–16.

[48] S. Van Dongen, "Graph clustering via a discrete uncoupling process," *SIAM Journal on Matrix Analysis and Applications*, pp. 121–141, 2008.

[49] G. Kollios, M. Potamias, and E. Terzi, "Clustering large probabilistic graphs," *TKDE*, pp. 325–336, 2013.

[50] R. Shamir, R. Sharan, and D. Tsur, "Cluster graph modification problems," *Discrete Applied Mathematics*, vol. 144, no. 1, pp. 173–182, 2004.

[51] N. Ailon, M. Charikar, and A. Newman, "Aggregating inconsistent information: Ranking and clustering," *J. ACM*, vol. 55, no. 5, Nov. 2008.

[52] A. Gionis, H. Mannila, and P. Tsaparas, "Clustering aggregation," pp. 341–352, 2005.

[53] P. C. Mahalanobis, "On the generalized distance in statistics," *Proceedings of the National Institute of Sciences (Calcutta)*, vol. 2, pp. 49–55, 1936.

[54] D. S. Hochbaum and D. B. Shmoys, "A best possible heuristic for the k -center problem," *Mathematics of Operations Research*, vol. 10, no. 2, pp. 180–184, May 1985.

[55] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985.

[56] K. Han *et al.*, "Efficient and effective algorithms for clustering uncertain graphs," *Proc. VLDB Endow.*, vol. 12, no. 6, pp. 667–680, 2019.

[57] G. Baltsou, K. Christopoulos, and K. Tsichlas, "Local community detection: A survey," *IEEE Access*, vol. 10, pp. 110 701–110 726, 2022.

[58] Z. Halim and J. H. Khattak, "Density-based clustering of big probabilistic graphs," *Evolving Systems*, vol. 10, no. 3, pp. 333–350, 2019.

[59] Y.-X. Qiu *et al.*, "Efficient Structural Clustering on Probabilistic Graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 10, pp. 1954–1968, Oct. 2019.

[60] 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.

[61] V. A. Traag, L. Waltman, and N. J. van Eck, "From louvain to leiden: Guaranteeing well-connected communities," *Scientific Reports*, 2019.

[62] M. Hamann, E. Röhrs, and D. Wagner, "Local community detection based on small cliques," *Algorithms*, vol. 10, no. 3, 2017.

[63] J. Huang, H. Sun, Y. Liu, Q. Song, and T. Weninger, "Towards online multiresolution community detection in large-scale networks," *PLOS ONE*, vol. 6, no. 8, pp. 1–11, Aug. 2011.

[64] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *Proc. VLDB Endow.*, vol. 9, no. 4, pp. 276–287, Dec. 2015.

[65] D. Lacamera, *Embedded systems architecture: explore architectural concepts, pragmatic design patterns, and best practices to produce robust systems*, eng. Birmingham Mumbai: Packt Publishing, 2018.

[66] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating systems: three easy pieces*, eng. Erscheinungsort nicht ermittelbar: Arpaci-Dusseau Books, LLC, 2018.

[67] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Physical Reveiw E*, vol. 78, 4 Oct. 2008.

[68] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowledge and Information Systems*, vol. 42, no. 1, pp. 181–213, Oct. 2013.

[69] R. Rabbany and O. R. Zaiane, "Evaluation of community mining algorithms in the presence of attributes," in *Trends and Applications in Knowledge Discovery and Data Mining*, X.-L. Li, T. Cao, E.-P. Lim, Z.-H. Zhou, T.-B. Ho, and D. Cheung, Eds., Lecture Notes in Computer Science - Springer, Nov. 2015, pp. 152–163.

[70] L. Peel, D. B. Larremore, and A. Clauset, "The ground truth about metadata and community detection in networks," *Science Advances*, vol. 3, no. 5, e1602548, 2017.

[71] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" In *Proceedings of the 19th International Conference on World Wide Web*, 2010, pp. 591–600.

[72] M. Zafarmand, "Community detection and discovery in deterministic and uncertain networks," Available Online, M.S. thesis, University of Alberta, 2020.

[73] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of Anthropological Research*, pp. 452–473, 1977.

[74] D. Lusseau, "The emergent properties of a dolphin social network," *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 2003.

[75] J. Dahlin and P. Svenson, "A method for community detection in uncertain networks," in *EISIC*, 2011, pp. 155–162.

[76] L. S. Heath and A. A. Sioson, "Multimodal networks: Structure and operations," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 6, no. 2, pp. 321–332, 2009.