COMMUNITIES IN SOCIAL NETWORKS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Sucheta Soundarajan

May 2013

COMMUNITIES IN SOCIAL NETWORKS

Sucheta Soundarajan, Ph.D.

Cornell University 2013

Within the broad area of social network analysis research, the study of communities has become an important and popular topic. However, there is little consensus within the field regarding the structure of communities, and the research literature contains dozens of competing community detection algorithms and community evaluation metrics.

In this dissertation, we present several connected contributions, each related to the general theme of communities in social networks.

First, in order to motivate the study of communities in general, as well as the work later in this dissertation, we present an application of community detection methods to the link prediction problem, in which one attempts to predict which edges in an incomplete network dataset are most likely to exist in the complete network dataset. We demonstrate that use of community membership information can improve the accuracy of various simple link prediction methods, sometimes by a large margin.

Next, we examine the structure of "real" annotated communities and present a novel community detection method. In this chapter, we study real networks, each containing metadata that allow us to identify "real" communities (e.g., all graduate students in the same department). We study details of these communities' structures and, based on these results, create and evaluate an algorithm for finding overlapping communities in networks. We show that this method outperforms other state-of-the-art community detection methods.

Finally, we present two related sections. In the first of these two chapters, we describe the Community Structure Analysis Framework (CSAF), a machine-learning-based method for comparing and studying the structures and features of communities produced through different methods. The CSAF allows a practitioner to select a community detection method best suited for his or her application needs, and allows a researcher to better understand the behavior of different community detection algorithms. In the second of these chapters, we apply the CSAF to a variety of network datasets from different domains, and use it to obtain interesting results about the structures of communities identified algorithmically as well as through metadata annotation.

BIOGRAPHICAL SKETCH

Sucheta Soundarajan graduated with her PhD in Computer Science from Cornell University in 2013, working under the supervision of Dr. John Hopcroft. Her research interests lie in the areas of social network analysis and data mining.

She was born in Columbus, Ohio, and obtained her bachelor's degrees in Mathematics and Computer/Information Science at The Ohio State University in 2005. Her hobbies include cycling, reading, and volunteering at animal shelters.

This dissertation is dedicated to my family.

ACKNOWLEDGEMENTS

My deepest gratitude goes to my advisor and my family.

This dissertation would not have been possible without my advisor, John Hopcroft, and I thank him for his patience, guidance, support, and most importantly, his fundamental kindness. I also thank Robert Kleinberg, for being the best possible role model a young scientist could want, and Carla Gomes, for inspiring me to use my work for good.

I am also grateful to my father for instilling in me my love of mathematics and science, and to my mother and brother, for often believing in me more than I believed in myself.

I thank my husband for proofreading every one of my papers, for keeping me on track, and for his endless love and support, even from hundreds of miles away.

I thank my Cornell friends Lakshmi Ganesh, Ainur Yessenalina, and Bruno Abrahao for their advice and support.

I thank David Chen, Jonathan Kropko, and Mukta Tripathy for their unwavering, unconditional friendship.

Finally, I am grateful to the SPCA of Tompkins County, where I spent many happy hours volunteering, for helping to keep my life balanced and for showing me how truly rewarding service to others can be.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Over the past decade, computational social network analysis has become an increasingly relevant area of research, contributing tools and methods for understanding the intricate arrays of links and relationships that underlie complex systems ranging from genetic interactions within a body to the whole of human society itself. Although classically a social science area, the field has more recently attracted attention from mathematical scientists such as physicists, statisticians, and computer scientists, who have contributed powerful methods for analyzing large networks, with important consequences for fields as diverse as politics, epidemiology, and economics.

The field of social network analysis exists at the confluence of a myriad of often-unrelated scientific disciplines, such as physics, psychology, and business. Its history, however, can be traced back most strongly to the mathematical subfield of graph theory and the sociological study of human interactions.

Graph theory as a branch of mathematical study dates back several centuries to Euler's study of the seven bridges of Königsberg problem (which asked whether one could trace a route over several landmasses and rivers using every bridge exactly one time [16]), but the term "graph" was not coined until much later by James Joseph Sylvester in an 1878 *Nature* paper [23].

At its core, graph theory is the formal study of relationships between objects. A *graph* or *network* is a mathematical structure containing *nodes* or *vertices* (objects) connected by *edges* or *links* (relationships). A graph is usefully represented as an $n \times n$ square *adjacency matrix* A, where n is the number of nodes, and A_{ij} is

1 (or some other non-zero value) if nodes i and j are joined by an edge (that is, if they interact), and 0 if they are not. A graph may be weighted or unweighted; in the simpler unweighted model, every edge is equally strong, whereas in the weighted model, different edges may have different strengths (and correspondingly different values in the adjacency matrix). Visually, a graph is typically depicted as a set of dots, representing nodes, connected by lines, representing edges.

Graph theoreticians are interested in the formal study of such structures, and classically have asked questions such as: How many colors are needed to color a map of a country so that no two adjacent territories are the same color? How can one select a minimum-weight subset of edges such that every node in the graph is adjacent to at least one of the edges in the subset? What is a fast algorithm that allows us to efficiently route resources from one node in the graph to another node in the graph?

Researchers in this area are additionally interested in models for producing graphs. In the highly influential Erdös-Rényi random graph model, one sets parameters n and p, which are used to produce a so-called $G(n, p)$ graph containing n nodes, in which the probability of any two nodes being connected is given by p.

While abstract graph theory is of great theoretical interest, social scientists have tended to be more interested in the structures and characteristics of real networks, such as societies, organizations, or families. French sociologist Émile Durkheim viewed sociology as the study of social facts: those values and social structures that exist outside of any single individual [14]. Soon after, sociologist Georg Simmel examined society through the associations of individuals with one another [54].

In 1967, psychologist Stanley Milgram performed his famous “Small World” experiment, in which individuals living in Nebraska were given a package, and were asked to forward it closer to a “target” individual in Boston [62]. At each step of the experiment, individuals with a package were only allowed to send it to someone that they personally knew; that recipient then sent the package to someone that they knew, and so on, until the “target” was reached. Through this experiment, Milgram attempted to identify the “distance” between two random individuals in a network. Soon after, sociologist Mark Granovetter published the extremely influential paper “The Strength of Weak Ties,” demonstrating that individuals often find jobs through “weak” ties (or casual acquaintances), and arguing that society is held together through both strong and weak bonds [22].

More recently, the proliferation of computer and internet technologies have resulted in vast quantities of network data of all sorts. Researchers in the area of social network analysis are interested in the same types of questions that sociologists have studied before, but the large amount of available data has required a fundamental shift in the methods and tools used to address these questions (e.g., from individual interviews to automated algorithms). As such, researchers have adapted formal techniques from graph theory for use on real data.

Real network data differs in several important ways from the abstract models typically studied by graph theorists. Most relevant to the topics in this dissertation, real networks tend to demonstrate a great deal of clustering; that is, unlike the random Erdös-Rényi graph model described earlier, we observe that individuals in real networks often group together into distinct clusters or communities. One important area of network research thus deals with identifying and studying communities in networks [17, 20]. Although this topic has its roots in the classic mathematical problems of graph clustering, current research has tended to focus on the structure and characteristics of "real" communities.

Research into communities in social networks is valuable for several reasons, both theoretical and applied. From a social scientific perspective, formally understanding the nature and characteristics of communities can give researchers insight into the behavior of individuals within a society: for example, one might use community analysis methods to study the tendencies of individuals to self-organize within an unstructured business environment or determine the extent to which network effects lead to the ostracization of children from a school group. Within the area of social network analysis itself, the study of communities can be used within larger projects to understand the evolution of networks or calculate the centrality of individual nodes. Commercially, community analysis is also of great value, with potential applications in tasks such as network visualization or advertisement targeting.

Traditionally, researchers have approached the problem of identifying communities by first creating mathematical definitions of what real communities *ought* to look like, and then designing algorithms to identify sets of nodes that match these descriptions. While this approach has advantages, it is typically not clear whether a particular mathematical definition of "community" is correct or appropriate.

Many such mathematical definitions are based in the principle that a community ought to contain elements that are tightly connected to one another and poorly connected to the rest of the network. In the global human social network, for example, such a community might correspond to an isolated island village, in which individuals know nearly everybody else on the island but few people off the island. Examples of such definitions include modularity and conductance, both of which consider a set of nodes with high internal connectivity and low external connectivity to be a good community [48, 29].

However, although such metrics are commonly used in applications- for instance, the professional networking website LinkedIn uses the Louvain method for modularity optimization [7] for the purposes of community visualization [63]-, they can suffer from serious drawbacks. Modularity in particular is known to suffer from a "resolution limit" that causes poor performance on large networks [18]. Thus, despite the popularity of certain algorithms within various application arenas, there is little consensus within the social network analysis research community as to which, if any, of these algorithms actually produce reasonable results.

Much research effort has been exerted in creating new community detection methods and corresponding mathematical formalizations of "community" ([17], for instance, describes at least 20 broad *categories* of community detection methods, each containing many individual algorithms). However, comparatively little time has been spent on characterizing and understanding the structures of real communities, in part because determining whether a community is "real" can be a difficult task.

1.1 Community Evaluation Metrics and Community Detection Algorithms

Communities can be understood and evaluated through the use of various metrics, and can be identified through the application of community detection algorithms. A metric is simply a mathematical evaluation of whether a given set of nodes is a good community: for example, as discussed earlier, several popular community metrics are based on the general intuition that a good community is internally well-connected and externally poorly-connected. These metrics are typically mathematical formalizations based on human intuitions about what a good community should be, and there is very little consensus within the research community as to which metrics ought to be used. In some cases, one may have an a priori belief that a metric is 'good', but directly optimizing for that metric is computationally intractable. In such cases, one could develop heuristic algorithms to optimize for the metric, or one may empirically demonstrate that some algorithm (which may not have been intended to optimize for the metric) tends to produces communities that score well when evaluated by the metric. For a given metric, there may thus be any number of algorithms intended to find sets that optimize for that metric; however, not all community detection algorithms are based upon a metric.

In real networks, different edges are typically not equally important. For example, in a social network, a connection between two siblings is likely stronger than a connection between two casual acquaintances. There are community metrics and community detection algorithms for both weighted networks, in which edges may have different strengths, and unweighted networks, in which all edges are viewed as equal. Many algorithms for identifying communities in weighted graphs are based on algorithms for finding communities in unweighted graphs, but will assign greater importance to edges with heavy weights. Because the datasets that we use in this paper do not generally have edge weight information, we consider only versions of community evaluation metrics and community detection algorithms that are appropriate for unweighted networks, where every edge is considered equal. However, many of the tools and methods that we discuss in this dissertation can be trivially modified for use on weighted networks.

In this section, we begin by describing two commonly used metrics. We then discuss several community detection algorithms, some of which are based on metrics and others of which are not.

1.1.1 Community Metrics: Conductance and Modularity

Two metrics that epitomize the concept that good communities are dense, isolated sets are conductance and modularity [48, 29]. Conductance, the simpler of these two metrics, measures the ratio of the number of edges leaving a community to the number of edges incident to that community. That is, for a network N with nodes V and adjacency matrix A (with elements a_{ij}), the conductance $C(S)$ of a set S is given by:

$$C(S) = \frac{\sum\limits_{i \in S, j \in \overline{S}} a_{ij}}{\sum\limits_{i \in S, j \in V} a_{ij}}. \tag{1.1}$$

A set of nodes with low conductance has few edges leaving the community, and many edges within the community, and so is a good community, whereas a set of nodes with high conductance is a weak community. Note that conductance is a function of a single community within a larger network, rather than an evaluation of a set of communities. Optimizing for conductance is computationally intractable; however, there exist heuristic algorithms for finding low-conductance cuts [41].

Unlike conductance, modularity is a function of a partitioning of a network's nodes. The modularity of a partition is defined as follows: Define m to be the number of edges in the network, $A_{i,j}$ as the number of edges between nodes i and j, k_i as the degree of node i, and $\delta(i, j)$ as 0 if i and j are in different parts of the partition and 1 if they are in the same part [48]. Then the modularity Q of a partition is:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(i, j). \tag{1.2}$$

This represents the number of edges within a set as compared to the number of edges expected in that set had the edges been distributed at random: a set with many in-links and few out-links is unexpected in a random graph, and will thus contribute heavily to the total modularity of the partition.

1.1.2 Community Detection Algorithms

Community detection algorithms can be very broadly categorized into methods that partition the network into disjoint sets and methods that find overlapping communities. Algorithms of the former type tend to have roots in classical problems of graph clustering, whereas methods of the latter type are more firmly grounded in the sociological intuition that real communities in real networks are likely not disjoint (e.g., [44]). Of the algorithms discussed below, Greedy Modularity Optimization and Infomap find a network partitoning, while Clique Percolation, Link Communities, and OSLOM find overlapping communities.

Greedy Modularity Optimization

Although it is computationally intractable to find a modularity-maximizing partitioning, several heuristic algorithms exist for this task. In this dissertation, we typically use the popular Louvain method for greedy modularity optimization [7]. In this algorithm, modularity is first optimized locally by grouping together nodes into small communities. These small communities are then grouped together into larger and larger communities, until a maximum modularity value is attained.

Infomap

Many community detection methods are based on random walks [17]. These methods are typically founded in the intuition that a random walk on a network will tend to mostly stay within a community. One algorithm illustrating this general principle is the Infomap partitioning algorithm of Rosvall and Bergstrom, which views a network as an analogue to a geographical map, and the problem of partitioning a network into communities as similar to the problem of deciding how to draw a map [52]. Rosvall and Bergstrom describe random walks along the network with a two-level encoding scheme in which each node is described both by its cluster name as well as its own (relatively short) local name within that cluster. Nodes in different clusters may share short local names (e.g., many cities have a “Main St.”, but no city has more than one “Main St.”).

Given a labeling, a random walk can be described in the following way: When the walk enters a node in a different cluster, then the name of the cluster is written, as well as the name of the node within the cluster. When the walk stays within the same cluster, then only the local name of the node is written. The goal of the Infomap partitioning algorithm is to identify clusters so as to minimize the expected length of a random walk’s description. Intuitively, this is accomplished by grouping together nodes that often appear close together in random walks into appropriately sized clusters.

To identify a clustering, the Infomap algorithm uses a greedy algorithm. Initially, each node is placed in its own community, and communities are merged together in such a way as to greedily minimize the expected length of a random walk’s description. Simulated annealing is then used to improve this result. Lancichinetti and Fortunato evaluated several algorithms and networks and concluded that Infomap was the most reliable of the methods evaluated [33].

Clique Percolation

One common method for locating overlapping communities is the Clique Percolation method [49]. For a specified k, this method first locates all k-cliques in the network and then “rolls” together adjacent cliques. Two k-cliques are considered adjacent if they share $k - 1$ nodes. A community is formed by beginning with one k-clique, adding all adjacent k-cliques, then adding all k-cliques adjacent to those added in the last step, and so on, until no further growth is possible. Because each node may appear in multiple cliques, this method can produce overlapping communities. This algorithm is implemented by CFinder, a freely available software package [3].

Clique Percolation is one of the earliest methods for identifying overlapping communities, and so has been highly influential. However, it suffers from several major drawbacks. First, finding and rolling together cliques in a network can be impractically slow, particularly on dense networks. Second, it is not clear how one can optimally select k, or even that a single value of k is appropriate for the entire network (e.g., a higher value may be more appropriate for dense sections of the network). Finally, the algorithm is very sensitive to missing data: even one missing edge is sufficient to break a clique.

Link Communities

A more recent method for finding overlapping communities is the Link Communities method [4]. This method is based on hierarchical clustering; however, instead of clustering nodes, it clusters edges. For a network G, this method creates a new network H in which every node in H represents an edge from G. Two nodes in H are linked by an edge if their associated edges are adjacent in G. An edge between two nodes in H is weighted according to the similarity between the associated edges in G, defined as the Jaccard similarity between the neighborhoods of their unshared nodes [28].

The algorithm then uses single-linkage hierarchical clustering to identify "link communities" in H, stopping when a maximum network partition density, defined as the average partition density D_C for all communities C, is achieved. The partition density D_C for a community C containing m links and n nodes is defined as follows:

$$D_C = \frac{m - (n-1)}{n(n-1)(n-2)}. \tag{1.3}$$

Note that because a node can be represented by multiple edges, it can appear in multiple communities.

OSLOM

OSLOM ("Order Statistics Local Optimization Method") is a clustering method intended to detect overlapping, hierarchical community structure and distinguish meaningful community structure from artifical communities that occur even in random networks [34].

The OSLOM procedure consists of three main parts: first, finding significant clusters; second, analyzing the resulting clusters to determine whether any should be merged or split; and third, identifying community hierarchies by repeating the analysis on a new network representing the clusters already detected.

To identify one cluster, the algorithm initially begins with a randomly selected node as its own cluster, and then adds significant neighbors of that cluster through a stochastic process that locates those nodes that have more edges into the cluster than would be expected in a random network. A clean-up procedure is then applied to the cluster, in which significant nodes are added to and insigificant nodes are removed from the cluster. Once clusters have been identified, the algorithm then considers merging or splitting the clusters. Next, the algorithm forms a new network in which the nodes represent clusters found in the earlier steps. The process is then repeated on this network, and so on, to produce a hierarchy of communities.

1.2 Organization

This dissertation contains four main contributions loosely connected by the common theme of community structure in networks.

First, to motivate the importance of community research, we present an application of community detection methods. We consider the link-prediction problem, in which one attempts to predict which links in incomplete network data are most likely to exist in the full data. This problem is relevant in fields such as genetics, where interactions between genes are typically determined experimentally. In such cases, identification of likely genetic interactions could assist researchers in their experiments, leading to savings of time and resources.

In the following section, we use external network annotation to identify "real" or "annotated" communities in networks, such as graduate students in the same department or books written by the same author. We examine properties of these annotated communities, and based on our conclusions, describe a novel algorithm template for finding overlapping communities. We demonstrate that this method outperforms other state-of-the-art methods when evaluated on the task of correctly identifying annotated communities.

We conclude by presenting a pair of contributions, each also relating to the structure of communities.

In the first of these two contributions, we present a three-part machine-learning-based framework for comparing and contrasting the structures of communities identified by different methods (e.g., community detection methods or external annotation). This framework is suitable for researchers interested in obtaining a better understanding of community detection methods, as well as for practitioners wishing to choose a suitable community detection method for a given application.

Finally, we use this framework to compare the structures of annotated communities to the structures of communities produced by various community detection methods. We draw several interesting conclusions, including the result that the class of annotated communities has a cohesive structure, but that the structure is not adequately captured by any of the community detection algorithms that we considered.

CHAPTER 2

DATASETS AND ANNOTATED COMMUNITIES

Throughout this dissertation, we analyze many networks of various types, some of which tag individual nodes with metadata. Some of these networks are originally directed or weighted, but we convert each network into an undirected, unweighted graph by simply removing directions or weights on edges.

Because there are so many different mathematical formulations of “community”, portions of this dissertation are concerned with characterizing or identifying “real” communities. To this end, many of the network datasets that we consider contain metadata that allow us to identify “annotated” communities. For example, one network that we consider is a portion of the Facebook network for graduate students at a university. For this network, we use the metadata to identify students that are in the same department. One example of an annotated community may be the set of all students within the Computer Science department. Another network that we consider is the Amazon product co-purchasing network. Here, each node is an product sold by Amazon.com, and an example of an annotated community is the set of all books written by the same author.

Study of annotated communities is important for several reasons. First, examination of the structure of annotated communities is naturally a valuable scientific topic in its own right, of interest to social scientists from many disciplines. Second, a better understanding of real communities helps lead to the development of better community detection algorithms (and thus improved performance in various applications) by both suggesting new mathematical formulations of community, as well as by allowing more objective evaluation of proposed community detection algorithms.

While algorithms are typically evaluated by measuring how well their output satisfies some mathematical formulation, such evaluation methods are naturally subjective (e.g., when output communities are evaluated by their conductance scores, an algorithm intended to optimize for conductance will outperform an algorithm intended to optimize for modularity, but this does not necessarily demonstrate that the former algorithm is "better"). In contrast, the existence of meaningful, structured "real" communities allows for an objective ground truth that can be used to compare algorithms of fundamentally different natures.

In general, we identify "annotated communities" by grouping together nodes with the same annotation. In all the networks that we consider, a node may appear in multiple communities. We consider only those annotated communities that contain connected components with at least ten nodes. If some annotated community contains multiple components that are of size ten or larger, we consider each to be a separate community. In this section, we describe each dataset and some of its properties.

2.1 Networks with Annotation

Amazon is a product co-purchasing network from Amazon.com [36]. Each node represents a book, and an edge exists between two nodes if one was frequently purchased with the other. The network contains $270,347$ nodes and $741,142$ edges. For each item in this network, Amazon.com provides several product categories, such as "Chemistry Textbooks" or "Spy Thrillers". We use these annotations to create a set of 9474 communities.

HS , **DM** , and **SC** are, respectively, biological networks describing protein-protein interactions for *H. Sapiens* (human), *D. Melanogaster* (a fruit fly species), and *S. Cerevisiae* (a type of yeast) [50, 55]. In these networks, a node represents a protein, and two nodes are connected if their associated proteins are known to interact with one another. HS contains $10,298$ nodes and $54,655$ edges, DM contains $15,326$ nodes and $486,970$ edges, and SC contains 5523 nodes and $82,656$ edges. Some proteins (though not all) are annotated with one or more gene ontology IDs describing the known function or functions that the protein serves. We use these gene ontology values to identify communities. HS contains 70 communities, DM contains 56, and SC contains 77.

Grad and **Undergrad (Ugrad)** are, respectively, sections of the Facebook network that correspond to graduate and undergraduate students at Rice University [45]. Grad contains 503 nodes and 3256 edges, and Undergrad contains 1220 nodes and $43,208$ edges. For each graduate student, we are given their anonymized department membership, college membership, and year. For each undergraduate student, we are given major, dormitory of residence, and year. We use this information to identify 24 communities in Grad and 41 communities in Ugrad.

Manu is a small network describing interactions of employees at a manufacturing plant [13]. Two workers are linked if one of them reported that he or she spoke to the other at least "somewhat infrequently". Manu contains 77 nodes and 705 edges. Using employment metadata for each worker, describing office location (city), length of time employed, and "organizational level" (e.g., "Local Department Manager"), we identify 10 communities.

DBLP is a computer science collaboration network, in which each node represents a published computer scientist, and two nodes are linked if the corresponding researchers have written at least one paper together. This network contains $632,203$ nodes and $2,391,252$ edges. We identify annotated communities in DBLP by grouping together researchers who have participated in the same publication venue (e.g., a conference or journal). We thus identify $10,595$ annotated communities in DBLP.

LJ1 and **LJ2** are two portions of the blogging website LiveJournal [6]. In these networks, each node represents a user of the LiveJournal website, and users explicitly declare friendship links. This network contains several million users, and as such, is too large for many of the community detection algorithms that we consider. We thus consider two $500,000$-node subsets of the network, each obtained by performing a breadth-first search at different locations in the network. LJ1 contains $10,736,588$ edges, and LJ2 contains $10,640,429$ edges. In addition to explicitly declaring friendship links, users may also explicitly declare membership in various communities. These communities may be based on interests (e.g., cooking), location, or other shared features. In LJ1, we identified $29,955$ annotated communities, and in LJ2, we identified $39,598$ annotated communities.

2.2 Networks without Annotation

In portions of this dissertation, we also consider networks without annotation.

HEP is a collaboration network drawn from research papers at the arxiv.com paper repository in the High Energy Physics- Theory category [40]. It contains 9877 nodes and $25,988$ edges.

Rel is a collaboration network drawn from papers at arxiv.com in the General Relativity and Quantum Cosmology category [40]. It contains 5242 nodes and $14,496$ edges.

Email is an e-mail network from the University Rovira i Virgili in Spain [24]. Email is a directed network, and we convert it to an undirected network by changing all arcs into edges. It contains 1133 nodes and 5452 edges.

Wiki is a network in which nodes represent Wikipedia users, and an edge between two nodes represents one node voting for or against promoting the other to an administrator position [39, 38]. It contains 7115 nodes and $100,762$ edges.

Word is an associative thesaurus created experimentally [32]. Researchers showed a few words to people, and the subjects replied with the first word that came to mind. These words were then added to the set, and the process was repeated, and so on. This is a directed network, and we convert it to an undirected network by changing all arcs into edges. This network contains $23,219$ nodes and $305,500$ edges.

CHAPTER 3

LINK PREDICTION: AN APPLICATION OF COMMUNITY DETECTION METHODS

In recent years, network analysis has become an increasingly popular topic for computer science researchers. However, much of the available network data is incomplete. For example, in a network representing interactions between genes in some species, links are typically determined experimentally, and so the known links may represent fewer than 1% of the actual links [60, 5]. Because of this, researchers sometimes wish to know which pairs of nodes that are not connected in the known network are likely to be connected in the actual network. Such knowledge is useful in cases like a gene interaction network, because it can suggest which experiments a biologist ought to perform to identify existing interactions. It is also useful for researchers designing or applying network algorithms, as such algorithms may perform more accurately when more links are known.

Algorithms for the link prediction problem typically compute the "similarity" between two nodes, with the assumption that nodes that are highly similar are more likely to be connected than those that are dissimilar [42]. The research question, then, lies in the question of how best to define "similarity." Simple measures consider easy-to-compute factors like the number of neighbors shared between two nodes, whereas more complex definitions partition the network into groups, and then determine the probability that two nodes are connected based on the group memberships of those nodes [10, 66]. Methods of the latter type can be more accurate than those of the former type; however, they typically run very slowly and may only be practical for small networks [42].

In this chapter, we consider several simple methods that use local information to predict the existence of a link between two nodes, but then supplement this local information with community membership information. For example, if two nodes, as well as some of their shared neighbors, are all in communities together, then we may infer that a link between them is more likely than if they and their shared neighbors are in different communities. We test our methods on 10 datasets from a variety of domains, ranging from scientific collaboration networks to friendship networks to gene interaction networks, and show that using community information often increases the precision accuracy of the results.

The remainder of this chapter is organized as follows: First, we discuss relevant background, beginning with a discussion of existing local similarity-based link prediction methods. Next, we describe the experiments we perform and list the datasets used in these experiments. After that, we discuss the results of these experiments, and finally conclude with suggestions for future work.[1]

3.1 Background

3.1.1 Local Similarity Measures

We consider 5 types of local similarity measures. Throughout this paper, we refer to these metrics as "base metrics", as they provide the foundation for our "enhanced metrics" that incorporate community information. For a vertex v, let $\Gamma(v)$ be the set of all neighbors of v, and let $d(v)$ be the degree of v. Then for nodes

[1]The work in this chapter originally appeared in [57].

a and *b*, we use the following base metrics:

- Common Neighbors: $|\Gamma(a) \cap \Gamma(b)|$, the number of neighbors shared by both *a* and *b*.
- Jaccard Similarity: $\frac{|\Gamma(a) \cap \Gamma(b)|}{|\Gamma(a) \cup \Gamma(b)|}$, the number of vertices adjacent to both *a* and *b* normalized by the number of vertices adjacent to either *a* or *b* [28].
- Sorensen Similarity: $\frac{|\Gamma(a) \cap \Gamma(b)|}{d(a)+d(b)}$, the number of vertices adjacent to both *a* and *b* normalized by the sum of the degrees of *a* and *b* [56].
- Resource Allocation: $\sum_{c \in \Gamma(a) \cap \Gamma(b)} \frac{1}{d(c)}$, the sum of the inverses of the degrees of vertices adjacent to both *a* and *b*. Intuitively, this is similar to the Common Neighbors measurement, but vertices with higher degree are worth less than those with low degree, because it is assumed that a high degree vertex connected to both *a* and *b* is less meaningful than a low degree vertex connected to both *a* and *b* [67].
- Leicht-Holme-Newman: $\frac{|\Gamma(a) \cap \Gamma(b)|}{d(a) \times d(b)}$, the number of vertices adjacent to both *a* and *b* normalized by the product of the degrees of *a* and *b* [35].

The Common Neighbors metric has been used in collaboration networks, where it has been shown that individuals who have collaborated with many of the same people are more likely to collaborate together in the future [46]. The Jaccard Similarity, Sorensen, and Leicht-Holme-Newman metrics use the same principle, but normalize this value, because if *a* and *b* are of high degree, a large number of shared neighbors is less meaningful than if *a* and *b* are of low degree. The Sorensen Similarity metric is used primarily in ecological applications, but we include it here for comparison. The Resource Allocation index is based on the principles of distributing resources along the edges of a network. If *a* has some unit of resource and distributes it equally to its neighbors shared by b, and each of those neighbors then allocates its portion of the resources equally between its neighbors, then the amount of resource that b receives is measured by the Resource Allocation index. Over a variety of networks, the Resource Allocation index has been shown to generally perform better than the other measures mentioned here [42].

3.1.2 Evaluating a Link Prediction Method

The success of a link prediction algorithm on a network N can be measured by two methods: precision and area under the receiver operating characteristic curve (AUROC) [42]. To calculate these, one can perform 10-fold cross validation. For each fold, 10% of the existing links are withheld from the network to create a new network N'. This is done 10 times, and each time, a different 10% of the links are withheld. This ensures that each link is withheld exactly once, so all links are present in the training data and the test data an equal number of times.

To measure precision, one uses N' to identify the n links that are most likely to exist in the full network, and then calculates the fraction of these n links that are present in the withheld 10% of links.

When evaluating precision scores in this paper, it is important to note that many of the network datasets are incomplete (even when none of the known links are withheld). Consider, for example, a biological network in which protein interactions are identified experimentally, and only 5% of the interactions have been identified. A perfect link prediction algorithm, then, might receive a precision score of only 5%: even if all of its predictions are actually correct, only 5% of the predicted links have been confirmed and count toward the precision score. Thus, while one should compare precision scores of different algorithms against one another on the same network, they should not be compared across datasets or against an absolute standard.

To measure AUROC, one samples m pairs of edges that are not in N', where one edge e_0 in the pair is present in the list of withheld edges and the other edge e_1 is not in this list. The link prediction method is used to score each pair by determining which of e_0 or e_1 is more likely to be present in the full network. If the method determines that e_0 is more likely than e_1 to be in the full network, then the pair is assigned a score of 1. If the method determines that e_0 and e_1 are equally likely to be in the network, then the pair gets a score of 0.5, and if the method determines that e_1 is more likely than e_0 to be in the network, then the pair gets a score of 0. To estimate the AUROC, one averages these scores over all sampled pairs.

3.2 Datasets

In this section, we use networks Amazon, Grad, Undergrad, HS, SC, Wiki, Word, Rel, HEP, and Email. We do not consider any metadata annotations, but simply use the raw link structure.

3.3 Evaluation of Base Metrics

For each network N listed in 3.2, we perform experiments using 10-fold cross validation. We partition N's links into 10 equal sized sets. We then perform 10 rounds of experiments. In each round, each such set is used as test data in one round, while the remaining 90% of the links are used as training data. For round i, let N_i denote the network defined by the links in the training data, and let T_i denote the set of links constituting the test data.

Then, using each of the base metrics described earlier, we identify the n most likely links that are not present in N_i, where $n = \frac{|T_i|}{10}$. We then calculate how many of the n found links are actually in T_i. This value, averaged over all 10 folds, is the precision of the metric. As before, we caution that the precision scores not be compared across networks or to an absolute standard. Many of the networks, such as the biological networks in which links are experimentally determined, are incomplete, even when links are not withheld. For such networks, even a perfect link prediction method could get a low precision score because although its predictions are all correct, the link does not exist in the known (incomplete) network data. Thus, a low precision score should not be taken as an indication that a method is objectively "bad"; rather, the precision scores should be used to compare algorithm accuracy.

Table 3.1 shows the results of evaluating each of the base metrics on each of the networks using 10-fold cross validation. For each network, the best performing metric has been bolded. Two metrics, Common Neighbors and Resource Allocation, are each the best performing metric for half of the networks, and the other metrics are not the best for any network.

Table 3.1: Precision of base metrics

	CN	Jacc	Sor	LHN	RA
Amazon	**0.3713**	0.0193	0.0193	0.0139	0.350
Grad	0.3713	0.5000	0.5000	0.0182	**0.7212**
Ugrad	0.5757	0.5870	0.5870	0.0322	**0.6889**
HS	**0.1110**	0.0113	0.0113	0.0007	0.0726
SC	**0.1944**	0.0306	0.0306	0.0000	0.0825
Email	**0.3509**	0.1000	0.1000	0.0018	0.3255
HEP	0.6988	0.6288	0.6288	0.0635	**0.9227**
Rel	0.9676	0.9828	0.9828	0.0717	**0.9903**
Word	0.1402	0.0013	0.0013	0.0000	**0.1471**
Wiki	**0.1773**	0.0001	0.0001	0.0000	0.1420

Caption: For every network, either base metric CN or base metric RA is the top performer.

3.4 Enhanced Link Prediction

In this section, we describe how we modify local similarity metrics, or “base metrics”, to account for community membership.

3.4.1 Modification of Local Similarity Measures

We believe that community membership information can provide valuable information for the link prediction problem. Consider the Common Neighbors similarity metric, one of the simplest local similarity measures. Under this metric, the similarity of two nodes *a* and *b* is defined by the number of neighbors that *a* and *b* have in common. Suppose that we are using this metric to analyze a friendship network.

Suppose that individual *c* knows both *a* and *b*, but *c* knows *a* from the community corresponding to some school, and *c* knows *b* from the community corresponding to some workplace. Suppose additionally that *a* and *b* both have a neighbor *d* in common, and *d* knows both *a* and *b* from the community corresponding to some sports team. When calculating the probability that *a* and *b* know one another, it is possible that the shared neighbor *d* should count more heavily towards this probability than the shared neighbor *c*, since *a* and *b* both know *d* from the same context. The fact that *a* and *b* are in at least one community together should also count towards this probability.

Similarly, suppose that *a* is adjacent to some vertex *c*, and *b* is not adjacent to *c*. Some metrics, such as the Jaccard Similarity metric, take into account all vertices adjacent to either *a* or *b*, not just those adjacent to both *a* and *b*. In such cases, we may wish to penalize more greatly those neighbors adjacent to only one of *a* or *b* that are in a community with both *a* and *b*. Intuitively, if *a*, *b*, and *c* are in a community together, and *c* is connected to *a* but not to *b*, then there is a greater sense that *c* *ought* to be connected to *b* than if *c* and *b* had been in different communities, and so the modified similarity measure receives a greater penalty for such cases.

Because Common Neighbors and Resource Allocation strongly outperformed the other local similarity-based metrics, we only consider these two measures for modification. In some metrics, we assign extra points to a pair of nodes *a* and *b* if they share neighbors in the same communities, or if *a* and *b* themselves are in the same communities.

For each method, we use the following shorthand: for nodes a and b, let $\Gamma(a)$ be the set of neighbors of a, $\Gamma(b)$ be the set of neighbors of b, $\Gamma(a, b)$ be the neighbors shared between a and b, $d(a)$ be the degree of a, and $d(b)$ be the degree of b.

We generate communities using the Louvain method for greedy modularity optimization, Infomap, and the Link Communities method. Because the Link Communities method generates communities of edges rather than nodes, we interpret its results in two ways: first, simply as a collection of overlapping communities of nodes, and second, as a partitioning of edges. Call the former method Link Communities- Node, and the latter method Link Communities- Edge. Call the former type of community a node community, and the latter type an edge community.

We consider a variety of modifications to the original local similarity measures. For a pair of nodes (a, b), most of these modifications either assign extra points for neighbors shared between a and b that are in some of the same communities as a and b, or assign extra points for all communities that a and b are both in, or both.

For each of the metrics described below, given a particular community detection method, let $C(n)$ be the set of node communities to which node n belongs. For Link Communities- Edge, let $C(n, m)$ be the set of edge communities to which edge (n, m) belongs. For greedy modularity optimization and Infomap, the size of $C(n)$ is 1, and for Link Communities- Edge, the size of $C(n, m)$ is 1. For Link Communities- Node, the size of $C(n)$ may be greater than 1.

Common Neighbors

We enhance the Common Neighbors (CN) metric in 5 ways. For nodes a and b, let $CN(a, b)$ be the number of common neighbors between a and b. The first three methods ($CN1, CN2, CN3$) can be computed using the node communities obtained from greedy modularity optimization, Infomap, and Link Communities-Node. The last two methods can be computed using the edge communities from Link Communities- Edge.

- Common Neighbors 1 (CN1): In this measure, $CN1(a, b)$ begins with the base score given by $CN(a, b)$, and then for every neighbor i shared by a and b, $CN1(a, b)$ receives an additional point for every community that a, b, and i are all in.

$$CN1(a, b) = CN(a, b) + \sum_{i \in \Gamma(a,b)} |C(i) \cap C(a) \cap C(b)| . \tag{3.1}$$

- Common Neighbors 2 (CN2): In this measure, $CN2(a, b)$ begins with the base score given by $CN(a, b)$, and then receives an additional point for every community that a and b are both a part of.

$$CN2(a, b) = CN(a, b) + |C(a) \cap C(b)| . \tag{3.2}$$

- Common Neighbors 3 (CN3): This measure is a combination of $CN1$ and $CN2$.

$$CN3(a, b) = CN1(a, b) + |C(a) \cap C(b)| . \tag{3.3}$$

- Common Neighbors Edge 1 (CNEdge1): In this measure, $CNEdge(a, b)$ begins with the base score given by $CN(a, b)$, and then for every neighbor i shared by both a and b, if the relationships between both a and i and b and

i fall into the same community, $CNEdge1(a, b)$ receives another point.

$$CNEdge1(a, b) = CN(a, b) + \sum_{i \in \Gamma(a,b)} |C(i, a) \cap C(i, b)| . \tag{3.4}$$

- Common Neighbors Edge 2 (CNEdge2): This is the same as $CNEdge1$, but with additional points for every community that contains both a and b.

$$CNEdge2(a, b) = CNEdge1(a, b) + |C(a) \cap C(b)| . \tag{3.5}$$

Resource Allocation

We enhance the Resource Allocation measure in 2 ways, the first computed using node communities from greedy modularity optimization, Infomap, or Link Communities- Node, and the second computed using edge communities from Link Communities- Edge.

- Resource Allocation 1 (RA1): $RA1(a, b)$ is the sum over all vertices $i \in \Gamma(a, b)$ of $\frac{1+|C(i) \cap C(a,b)|}{d(i)}$. This is similar to the original Resource Allocation definition, but we give extra weight to shared neighbors i that are in at least one community with both a and b, and weight i's contribution toward the total score by the number of communities that i shares with a and b.

$$RA1(a, b) = \sum_{i \in \Gamma(a,b)} \frac{1 + |C(i) \cap C(a) \cap C(b)|}{d(i)}. \tag{3.6}$$

- Resource Allocation Edge 1 (RAEdge1): This is similar to $RA1$, except we give extra weight to shared neighbors i such that (i, a) and (i, b) are in at least one edge community together.

$$RAEdge1(a, b) = \sum_{i \in \Gamma(a,b)} \frac{1 + |C(i, a) \cap C(i, b)|}{d(i)}. \tag{3.7}$$

3.5 Experimental Methodology

For each network N listed above, we perform experiments using 10-fold cross validation, as we did for the base metrics. We partition N's links into 10 equal sized sets. We then perform 10 rounds of experiments. In each round, each such set is used as test data in one round, while the remaining 90% of the links are used as training data. For round i, let N_i denote the network defined by the links in the training data, and let T_i denote the set of links constituting the test data.

For each round of experiments, we use only the training data to generate communities using Infomap, the Louvain method for greedy modularity optimization, and Link Communities (that is, the community detection algorithms are only given 90% of the known links).

Then, using each of the base and enhanced metrics described in Section 3.4.1, we follow the same cross-validation procedure as described in Section 3.3.

We also calculate the AUROC score. We select 1000 pairs of edges in which the first edge is present in T_i and the second edge is not present in either N_i or T_i, and then use each metric to score the two edges. Using this information and averaging over all 10 folds, we estimate the AUROC value.

3.6 Results

In this section, we discuss the modified local similarity metrics from Section 3.4.1. Our results show that the modified local similarity measures perform quite well in comparison to the base local similarity measures.

Table 3.2: Precision of base and enhanced metrics using Link Communities method

	SC	Ugrad	HS	HEP
CN	0.1944	0.5757	0.1110	0.6988
CN1	0.3746	0.6491	**0.1572**	0.7762
CN2	0.2441	0.6648	0.1343	0.7565
CN3	0.3733	0.6514	0.1422	0.7734
CNEdge1	3486	0.6671	0.1402	0.8038
RA	0.0825	0.6689	0.0726	0.9227
RA1	0.3724	**0.7007**	0.1111	**0.9281**
RAEdge1	**0.4707**	0.7144	0.1256	0.9277

Caption: One of RA1, RAEdge1, or CN1 is the top performer on each of these networks.

3.6.1 Modified Local Similarity Measures

Two main results emerge from our experiments on enhancing base local similarity metrics with community information: first, the best community information enhanced metrics outperform the best base metrics on precision, but the two types of metrics perform roughly the same when evaluated using the AUROC measure.

We see similar results when enhancing these base metrics with community information: the enhanced Jaccard, Sorensen, and LHN metrics do not perform the best on any network. To save space, we present only a few representative results. Table 3.2 contains the precision results obtained by evaluating each base and enhanced metric on the SC, Ugrad, HS, and HEP datasets using communities obtained from the Link Communities algorithm. Again, for every network, the best score (shown in bold) occurs with either $CN1$, $CNEdge1$, $RA1$. We see similar results for every other network except Grad, in which the best metric is the base Common Neighbors metric CN.

For the rest of this section, to save space, we thus only present results for CN, $CN1$ (and the related $CNEdge1$), RA, and $RA1$ (and the related $RAEdge1$).

We now present the CN, $CN1$, $CNEdge1$, RA, $RA1$, and $RAEdge1$ results for all networks using all community detection methods. Tables 3.3 and 3.4 contain the results for each network using communities found using the Link Communities (LC) method, the Louvain method (Mod), and Infomap (IM).

On every network except Grad, the best performing metric is one that incorporates community information. On 7 out of the 10 networks, some form of RA1 outperformed the other metrics, both base and enhanced (including ones not presented here), although it is not clear which community detection method is best. Additionally, regardless of choice of community detection algorithm, all of $CN1$, $CNEdge1$, $RA1$, and $RAEdge1$ outperform their corresponding base metrics on average, sometimes by a large factor.

Although for some networks, it appears that every metric does poorly, we again caution that this is not necessarily the case. For some networks (especially the biological networks, like HS) only a very small fraction of all links are known, and thus even a perfect link prediction would appear to have a low score. For other networks, such as Rel, it appears that the enhanced metric improves the base metric only slightly: for example, the precision of RA on Rel is 0.9903, and the precision of $RAEdge1$ is 0.9945. While this is only an increase

Table 3.3: Precision for base and enhanced Common Neighbors metrics

	CN	CN1(mod)	CN1(LC)	CN1(IM)	CNEdge1
Amazon	0.3713	0.3717	0.3220	0.3740	0.3326
Grad	0.5515	0.5455	0.5364	0.5364	0.5484
Ugrad	0.5757	0.6748	0.6491	0.6738	0.6671
HS	0.1110	0.1196	**0.1572**	0.1203	0.1402
SC	0.1944	0.2961	0.3746	0.2734	0.3486
Email	0.3509	0.3509	0.3291	0.3418	0.3691
HEP	0.6988	0.6831	0.7762	0.7031	0.8038
Rel	0.9676	0.9676	0.9676	0.9676	0.9676
Word	0.1402	0.1486	0.1015	0.1473	0.1101
Wiki	0.1772	**0.1951**	0.1437	0.1894	0.1515
Average	0.4139	0.4353	0.4357	0.4327	**0.4439**

Caption: Boldface indicates the metric that had the highest accuracy; if no column in a row is in boldface, then a RA-based metric was the top performer.

of 0.0042 in absolute terms, it covers nearly half of the distance from the base metric to a perfect score.

These results demonstrate that enhancing *CN* and *RA* with community information typically leads to an improvement in precision. We next consider how community information affects AUROC.

Table 3.4: Precision for base and enhanced Resource Allocation metrics

	RA	RA1(mod)	RA1(LC)	RA1(IM)	RAEdge1
Amazon	0.3507	0.3519	**0.4114**	0.3783	0.4097
Grad	**0.7212**	0.7000	0.7152	0.6879	0.7030
Ugrad	0.6889	0.7236	0.7007	**0.7241**	0.7144
HS	0.0726	0.0755	0.1111	0.0762	0.1256
SC	0.0825	0.1520	0.3724	0.1374	**0.4707**
Email	0.3255	0.3455	**0.3836**	0.3473	0.3764
HEP	0.9227	0.9196	**0.9281**	0.9204	0.9277
Rel	0.9903	0.9917	0.9917	0.9917	**0.9945**
Word	0.1471	**0.1490**	0.1276	0.1403	0.0940
Wiki	0.1420	0.1397	0.1459	0.1419	0.1187
Average	0.4443	0.4548	0.4888	0.4546	**0.4935**

Caption: Boldface indicates the metric that had the highest accuracy; if no column in a row is in boldface, then a CN-based metric was the top performer.

3.6.2 AUROC

When evaluating link prediction metrics through the AUROC score, we see in Table 3.5 that the enhanced metrics perform nearly identically to base metrics; their main advantage over base metrics occurs in the task of accurately predicting the most likely links (i.e., the task measured by precision). The strength of enhanced local similarity metrics, thus, lies in high precision scores.

Note that when we evaluate link prediction metrics through AUROC and precision, we are evaluating performance in fundamentally different ways. The

Table 3.5: AUROC for base and enhanced Metrics

	CN	CN1	CN2	CN3	RA	RA1
Amazon	0.8642	0.8669	0.8660	0.8712	0.8660	0.8641
Grad	0.9451	0.9450	0.9494	0.9488	0.9484	0.9476
Ugrad	0.9125	0.9142	0.9122	0.9108	0.9174	0.9216
HS	0.7709	0.7698	0.7847	0.7775	0.7719	0.7732
SC	0.8601	0.8525	0.8575	0.8537	0.8666	0.8749
Email	0.8427	0.8389	0.8459	0.8499	0.8481	0.8431
HEP	0.8988	0.9006	0.8999	0.9038	0.9026	0.9037
Rel	0.9202	0.9209	0.9219	0.9230	0.9219	0.9237
Word	0.7580	0.7597	0.7604	0.7568	0.7557	0.7555
Wiki	0.9264	0.9281	0.9242	0.9286	0.9342	0.9298

Caption: AUROC scores are roughly equal across all metrics.

AUROC score is useful for comparing metric performance over the set of all missing links, whereas when we calculate precision, we only consider those missing links that a metric considered very likely to exist in the full network. Thus, for tasks in which the ability of a metric to determine the most likely links is most important, enhanced local similarity measures are likely to be of great value.

3.7 Metric Selection

In nearly every case, the enhanced metrics outperformed the associated base metrics. In the cases where the enhanced metrics performed equally to or worse than the base metrics, it is possible that the community information was flawed. To evaluate the likelihood of this possibility, we re-calculate each metric using community information obtained by applying the community detection algorithms to the entire set of edges. That is, when identifying communities, we use all edges, rather than just the 90% of edges contained in the training set. All other portions of each metric, such as the number of shared neighbors, are still calculated using only the edges in the training set. Unsurprisingly, we see a fairly significant improvement in performance: even for network Grad, where the best metric was previously the base metric RA, the best metric is now RAEdge1. Naturally, we are not proposing this as a solution to the metric selection problem: in a real application, a practitioner certainly will not have access to the complete set of edges, and so must use inaccurate community information, but this experiment demonstrates the importance of correct community membership data.

In order to assist a practitioner in selecting an appropriate metric, we present the following two methods.

3.7.1 Metric Selection through Cross-Validation

In our original experiments, we performed 10-fold cross-validation. In each iteration, 10% of the network's edges were withheld for testing and the remaining

90% used for generating communities and making predictions. For each of these training sets containing 90% of the edges from the original network, we perform another level of 10-fold cross-validation by further dividing those edges into 10 sets of training and testing edges. We apply the link prediction methods to this second layer of cross-validation sets.

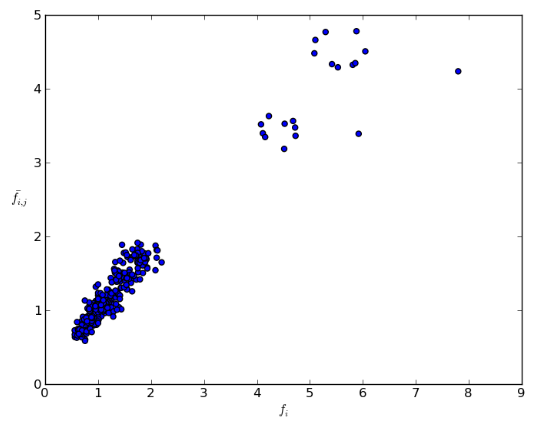
For example, consider network SC. In the previous section, we created 10 sets of training edges. Call these sets SC_1, SC_2, ..., SC_{10}. For each SC_i, metric $RAEdge1$ produces some fraction improvement f_i over metric RA when evaluated on the corresponding set of test edges (the ratio of the $RAEdge1$ score to the RA score is approximately 6 on average, but varies for each SC_i). For each SC_i, we create 10 more sets of training edges, $SC_{i,1}$, ..., $SC_{i,10}$. For each $SC_{i,j}$, metric $RAEdge1$ gives some fractional improvement $f_{i,j}$ over metric RA when tested on the corresponding set of test edges. For each SC_i, we calculate the average $\overline{f_{i,j}}$ over all values j of $f_{i,j}$.

We then plot f_i against $\overline{f_{i,j}}$ for all values i, all networks, and all enhanced metrics. Results are shown in Figure 3.1. The purpose of this experiment is to show that $\overline{f_{i,j}}$ is strongly related to f_i: thus, to select a metric for some network, one can observe that metric's cross-validation performance on subsets of the network.

We see that there is a fairly strong relationship between each f_i and $\overline{f_{i,j}}$. In particular, when f_i is extremely high (as in the case of network SC), $\overline{f_{i,j}}$ is also very high, and vice versa.

These results give guidance to users looking to apply these methods to real data. Although no metric is best for every network, one can simply perform

Figure 3.1: $\overline{f_{i,j}}$ vs. f_i



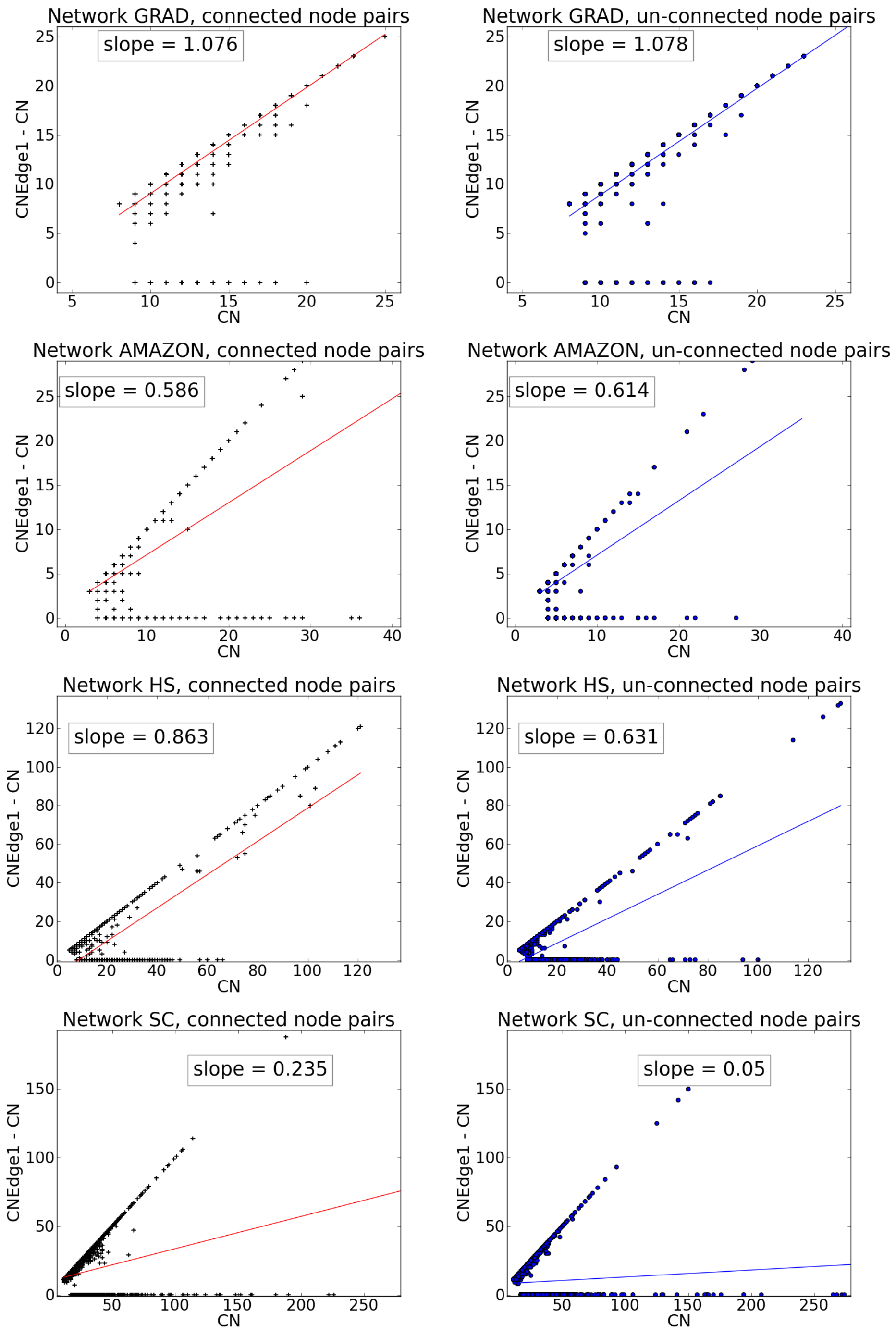
Caption: Cross-validation accuracies are a good predictor of actual accuracy.

cross-validation on that network to determine whether a particular metric is likely to be successful when applied to the complete data.

3.7.2 Metric Selection through Comparison of Existing Edges and Non-Edges

In addition to performing the cross-validation method of metric selection described above, one can also use the existing data to compare pairs of nodes that are connected and pairs of nodes that are not connected. In this experiment, for every training network, we generate a list of all pairs of nodes (u, v) that share at least one neighbor. For some of these pairs, u and v are connected, and for other pairs, they are not. For each pair in the list, we calculate the value of the $CN(u, v)$ and $CNEdge1(u, v)$ metrics. We then plot $CNEdge1(u, v) - CN(u, v)$ vs. $CN(u, v)$,

Figure 3.2: Plots of $CNEdge1 - CN$ vs. CN



Caption: The left columns contains values of $CNEdge1 - CN$ vs. CN for pairs of nodes that are connected, and the right column contains values for pairs of nodes that are not connected.

and determine whether the $CNEdge1(u, v) - CN(u, v)$ values differ significantly between pairs of nodes that are connected and not connected, but have the same $CN(u, v)$ value. Intuitively, if metric $CNEdge1$ is to be successful relative to metric CN, then if u and v are connected, we expect $CNEdge1(u, v) - CN(u, v)$ to be higher than if they are not connected.

On Grad and Amazon, $CNEdge1$ performed worse than CN, and on SC and HS, $CNEdge1$ outperformed CN. To save space, we present the plots for only these networks in Figure 3.2. In Figure 3.2, the left column contains the plots for connected pairs of nodes, and the right column contains the plots for un-connected pairs of nodes. We limit the range of CN and $CNEdge1 - CN$ values that we consider, and consider only those pairs of nodes that had high CN or $CNEdge1$ values, because these pairs of nodes are the ones that effect precision scores (as precision only considers the most likely edges). In this case, we consider the top e pairs of nodes as measured by either CN or $CNEdge1$, where e is the number of edges in the test set. For each plot, we calculate the best-fit line.

From this figure, we quickly reach several conclusions. First, for those networks (SC and HS) on which $CNEdge1$ outperformed CN, the slope of the best-fit line is significantly higher for the connected nodes than for the un-connected nodes (this is especially true for SC). This indicates that connected pairs of nodes tend to have much higher $CNEdge1$ values than un-connected nodes with the same CN value, and so $CNEdge1$ does a good job in discriminating between connected and un-connected node pairs. In contrast, on Grad and Amazon, the slopes of the best-fit lines are nearly identical. This strongly suggests that, for these two networks, $CNEdge1$ is unlikely to outperform CN (as is indeed the case).

Note also that for HS and SC, there are relatively few pairs of connected nodes that have very high CN values and low $CNEdge1 - CN$ values (along the x-axis). This is particularly apparent for HS, which has no pairs of connected nodes along the x-axis with CN greater than approximately 70. In contrast, there are many pairs of un-connected nodes in this location. Conversely, for Amazon and Grad, the plot containing connected pairs has many elements with high CN and $CNEdge1 - CN = 0$.

This method of comparing plots is a simple way to determine whether the incorporation of community information is likely to be useful, and unlike the earlier cross-validation method, does not require reapplication of community detection methods.

3.8 Conclusion and Future Work

We have considered the problem of link prediction in incomplete networks. Much of available network data is incomplete, and so researchers are interested in methods for predicting which edges are most likely to exist in the full network. Many approaches to this problem use local similarity metrics to determine which pairs of nodes are most similar, and thus most likely to have an edge between them. We consider several such metrics and enhance each one with community membership information.

Of the 5 base metrics, Common Neighbors and Resource Allocation were the most successful over all networks: each was the best performing network on 5 out of the 10 networks. We thus focused on these metrics and their modifications.

Averaged over all 10 networks, enhanced metrics $CN1$, $CNEdge1$, $RA1$, and $RAEdge1$ had higher precision scores than their corresponding base metrics, and similar AUROC scores. We thus conclude that the addition of community information to the Common Neighbors and Resource Allocation metrics can improve their performance on many types of networks. Although no single metric was best for every network, we showed that one can perform cross-validation to determine whether a particular method is likely to succeed on a given network.

Future directions for this problem might focus on other ways to determine which community detection method is best suited for a particular network. In our results, Link Communities- Edge seemed to be the most successful method, but results were mixed. This problem is naturally related to the problem of determining which community detection method is best suited for finding communities within a network, but it it not obvious that the two problems are the same. A related problem is that of determining which base local similarity metric is best for a particular network. Of the base local similarity metrics that we evaluated, Common Neighbors and Resource Allocation outperformed the others, but can one determine which metric to use based on features of the network?

CHAPTER 4

ON CHARACTERIZING AND IDENTIFYING COMMUNITIES IN NETWORKS

In the last chapter, we saw the need for reliable and accurate community detection algorithms. Traditionally, researchers have approached the community detection problem by first creating mathematical definitions of what real communities *ought* to look like, and then designing algorithms to identify sets that match these descriptions. While this approach has advantages, it is not always clear whether a particular mathematical definition of "community" is correct.

Many such mathematical definitions are based in the principle that a community ought to be "round," or well-connected throughout: for example, a good community might resemble a $G(n, p)$ graph within a larger network (with p large relative to the overall edge density of the network). Classic examples of such definitions include modularity and conductance, both of which reward a set of nodes for having high connectivity throughout the entire set.

Another type of mathematical definition is founded in the belief that communities are "long," or formed of many small groups that are individually well-connected, and while those small groups may be well-connected to one another, each individual node in a group may not be well-connected to the rest of the community. For example, the popular Clique Percolation algorithm [49] first identifies cliques of a certain size, and then "rolls" together adjacent cliques (those sharing all but one node) to find larger communities. While portions of such a community are certainly well-connected (as they are cliques), an individual node need not have any connection to more distant cliques.

Because there is little consensus about what real communities are, rather than creating and evaluating our method based on some mathematical criterion, we choose to design it and test it using real data. We use a collection of seven network datasets from varied domains, including social, product, and biological. Each of these networks contains some sort of external annotation that allows us to identify "annotated communities." For example, in a social network of students at a university, all students in the same department constitute one annotated community.

First, in order to gain insight into which of the two concepts of "community" is more realistic, we examine the annotated communities in detail. We demonstrate that annotated communities tend to be much "longer" than random graphs of the same size, and so conclude that the "long" model of communities as sets of small groups may better charaterize annotated communities than the "round" model of communities. We then decompose each annotated community into several constituent parts, and show that these parts tend to fit the "round" model much better than do the complete annotated communities.

Working with these principles, we create the Node Perception algorithm template for finding overlapping communities in networks. Our method is founded partly in the intuition that while individuals may belong to many different communities, a relationship between two individuals will generally fall solidly into one community. Given this, individuals in a network should be able to partition their neighbors into disjoint "sub-communities" that are portions of larger communities. For example, an individual person may be in many communities, such as her workplace, a university department at her school, an extended family, and so on. While she cannot name every individual in these communities, she can probably identify which of her acquaintances fall into each of these communities, and so can group her neighbors into sub-communities (e.g., “my co-workers”, “my classmates”, etc.). These sub-communities can be identified through use of a simple graph partitioning algorithm, or, in some cases, may be more accurately identified with available metadata (e.g., if several people frequently appear together in photographs). We then identify communities in a new network in which each node represents a sub-community. Each node from the original network can be represented by multiple sub-communities, so a node can appear in many different communities. Because a practitioner may choose how to identify sub-communities, how to create a network of sub-communities, and how to identify communities in that new network, this template is highly flexible and can easily be tuned to meet the user’s needs. To evaluate our method, we test how well it recovers the set of annotated communities. Because it is a flexible template, we consider several specific instances, and show that all of these instances outperform several other popular methods for identifying communities.

We finish with a brief discussion of how a practitioner might select a specific Node Perception implementation to suit his or her needs and based on the network features. We give several case studies, in which we consider features of actual networks, and show how modifications based on these features can further increase the performance of Node Perception.

Our work is novel in several important ways. Publications in the community detection arena typically present an algorithm and show that it is effective on some datasets, but often do not examine why a method is successful. Alternatively, they may present an algorithm and then give theoretical guarantees, with little justification for why a particular theoretical measure is best. Although many other popular algorithms are based on the general principle of joining together small, well-connected groups of nodes (including the recent DEMON algorithm [12], which is a specific instance of our method), to our knowledge our work is the first to propose an explanation based on real data for why such methods of community detection are successful. Additionally, we demonstrate that the general template is far more important than the specific implementation. Indeed, in most cases, each of the six implementations that we consider outperforms the other algorithms. To both practitioners and researchers, this conclusion is of greater value than demonstrating that one single algorithm outperforms other methods. To practitioners, these results are particularly useful because our method gives a user a great deal of flexibility, even allowing for the easy incorporation of information external to the structure of the network, such as in the case when some community memberships are known. Just as importantly, these results are valuable to researchers, because they give insight into the structure of annotated communities by demonstrating that the template itself, rather than specific implementations, is responsible for Node Perception's success.

This paper is organized as follows: first, we discuss work related to ours, including descriptions of other algorithms that we use for comparison. Next, we list each of the datasets used for evaluation. After that, we discuss our method in detail. We then compare the output from each algorithm to the annotated communities from the datasets. We find that averaged over seven datasets, our Node Perception methods outperform the other tested methods. We then discuss scalability and suitability of different Node Perception implementations, and consider several case studies, in which we use features of individual networks to select an appropriate Node Perception implementation. Finally, we discuss some directions for future work.[1]

4.1 Related Work

Many traditional community detection algorithms are based on the principle that a good community is a set of nodes that is well connected internally and mostly separated from the rest of the network. This has led to the formulation of measures such as modularity [48], which measures the number of edges within a community as compared to the expected number if the edges had been distributed randomly, and conductance [29], which measures the ratio of the number of edges within a set to the number of edges outgoing from the set. Such concepts have led to a diverse set of algorithms, which typically produce a partitioning of the network.

Algorithms for finding overlapping communities are also quite diverse. As with partitioning algorithms, some are intended to optimize some mathematical definition. Others, like Link Communities and Clique Percolation, are founded on the concept that communities are based on very small, local groups. Leskovec's Kronecker graph generative model is a recursive model of graph generation in which the structures of small portions of the graph resemble the way that those portions are connected to one another [37]. Some algorithms, such as described in [19], identify local sets that are similar to our "sub-communities," and then expand these "egomunities" into larger communities so that a particular mathematical feature of the communities is maximized.

[1]The work in this chapter originally appeared in [58].

Unlike many of these algorithms, our method is not based on a rigid mathematical optimization, and can easily be modified for various network features or to incorporate metadata. Clique Percolation and, in particular, Link Communities might also be considered templates, in the sense that they are easily modified. However, while these methods are superficially similar to our method in the sense that they join local groups of nodes together, they have an inflexible notion of local groups: Link Communities takes each edge to be its own local group, and Clique Percolation identifies cliques. In contrast, our method provides a flexible way for identifying such "sub-communities."

Most similar to our work is the recent DEMON algorithm [12], which can be viewed as a specific instance of our Node Perception template.

In this paper, we compare our method to several other algorithms, most of which were discussed in Section 1.1. We include the Louvain method for greedy modularity optimization and Infomap, two methods for partitioning the network, as well as the Link Communities, OSLOM, and Clique Percolation algorithms for identifying overlapping communities. In addition, we consider the DEMON algorithm for finding overlapping communities, which, as an instance of our template, we discuss in greater detail later in this chapter.

4.2 Datasets

In this chapter, we use the following network datasets and corresponding community annotations: student Facebook networks Grad and Undergrad; genetic interaction networks HS, SC, DM; product co-purchasing network Amazon; and employee interaction network Manu.

4.3 Community Structure

As discussed earlier, many classic conceptions of community structure are based on the belief that communities should be well-connected internally. Two such examples, both of which are popular and frequently used, are modularity and conductance. Observe that both of these metrics expect links throughout the community: that is, these metrics are useful for finding "round" communities that are well-connected through the entire community, much like a $G(n, p)$ graph [15]. Such communities would tend to have low diameter and low clustering coefficient.[2] The Infomap and OSLOM communities also fall into this general category.

In contrast, other algorithms, such as Clique Percolation, find "long" communities, in which there is no expectation that a node at one "end" of a community ought to be connected to a node at another "end." These communities may exhibit a higher clustering coefficient and higher diameter. For instance, consider a community found by Clique Percolation that was identified by connecting adjacent k-cliques $C_1, C_2, ..., C_r$, where two cliques are adjacent if they share $k - 1$ nodes. If there is an edge between some node in C_1 and some node in C_r, the Clique Percolation algorithm does not view this community as any stronger than if there is no such edge. Note that this model is not entirely the same as a model of community hierarchy, although in both models, communities consist of small groups. Consider, for example, a community found by Clique Percolation, which is clearly made of small groups, but may have have no deep hierarchical structure.

[2]We do not claim that every community found by a modularity- or conductance-optimizing algorithm will have this sort of structure, but rather that the metrics are based on the principle that communities ought to look like this.

At the extremes, this difference between "round" communities and "long" communities is illustrated by the contrast between a "round" clique, in which there must be connections throughout the entire set, and a "long" connected component, in which the connectedness may be very localized.

4.3.1 Community Roundness

Rather than define "round" and "long" in absolute terms, we view them as opposite ends of a spectrum. We define the "roundness" of a community as the ratio of its diameter to the diameter of a connected random Erdös-Rényi $G(n, M)$ graph with the same number of nodes and edges. A very "round" community has a low diameter relative to the random graph; a "long" community has a high diameter relative to the random graph. In this definition, an Erdös-Rényi $G(n, M)$ graph epitomizes the concept of a graph that is well-connected throughout. Note, however, that some graphs may be *"rounder"* than a $G(n, M)$ graph of the same size: for example, a star graph with a large number of nodes has diameter 2, while a $G(n, M)$ graph of the same size will likely have a much larger diameter. The "roundness" statistic of a community may thus take on any positive value, where small values indicate a "round" community and large values indicate a "long" community.

In this chapter, to estimate a community's roundness values, we perform the following procedure: for a given set with n nodes and M edges, we produce a random connected Erdös-Rényi $G(n, M)$ graph with the same number of nodes and edges. In each case, we make up to $10,000$ attempts to generate a connected random graph of the same size. If we were unable to generate a connected graph, we discarded that community from consideration. This typically occured when a graph had many nodes and relatively few edges. We then measure the ratio of the community's diameter to the diameter of the random graph.

4.3.2 Roundness Analysis

In order to determine whether the annotated communities from the networks listed in the previous section are "round" or "long," we calculate their roundness statistics. For each annotated community, we first identify a "core" of that community by iteratively eliminating all nodes with only one edge into the community. We perform this trimming process in order to ensure that our diameter calculations are not skewed by communities that are very weakly connected and have large diameters (e.g., communities with long path-graph-like structures on the fringe). We then calculate the roundness of the trimmed community.[3]. Note that this trimming procedure can only decrease a community's diameter, and thus its roundness statistic (that is, the original community can never be rounder than the trimmed community). If annotated communities are indeed "round," then we expect that, on average, their diameters will be roughly similar to the diameters of the corresponding random graphs. In contrast, if the annotated communities are "long," then we expect that they will have larger diameters than their corresponding random graphs.

The first row of Table 4.1 contains the results of these experiments. For each network, we present the average roundness of the annotated communities. For some networks, such as network HS, the diameters of the annotated communi-

[3]All network features in this chapter were calculated using the NetworkX software package for Python [25].

Table 4.1: Ratios of diameters of annotated communities, annotated community parts, and networks of parts to diameters of random graphs of the same size.

	Grad	Ugrad	HS	SC	DM	Amaz.	Manu
Ann. Community Diameter Ratios # nodes ≥ 50	1.34	1.29	1.11	1.17	1.21	1.16	1.23
Ann. Community Diameter Ratios # nodes ≥ 50	1.56	1.47	1.17	1.29	1.50	1.72	1.33
Ann. Community Diameter Ratios # nodes ≥ 75	1.58	1.54	1.13	1.32	1.50	1.84	–
Ann. Community Diameter Ratios # nodes ≥ 100	1.70	1.67	1.15	1.30	1.75	1.91	–
Community Parts Diameter Ratios	1.07	1.16	1.04	1.06	1.01	1.03	1.02
Community Parts Network Diam. Ratios	1.02	1.00	1.00	0.99	0.91	1.00	1.00

Caption: Annotated communities tend to not be "round"; this effect is magnified for larger communities. However, they are formed of parts that are "round", and these parts are connected to one another in a "round" way.

ties are only slightly larger than the diameters of the random graphs. For other networks, the difference is far more pronounced. These results suggest that the various models of "round" communities may not be well-suited for characterizing these annotated communities. Interestingly, for many networks, this difference becomes increasingly pronounced as we consider larger and larger communities: for example, in network DM, the average ratio was 1.50 when considering communities of size at least 50, but 1.75 when considering only communities of size at least 100.

In our next experiment, we decompose each annotated community into several parts, and examine whether each of these parts individually better fits a "round" model. To perform these decompositions, we use the Louvain method for greedy modularity optimization. This process gives us a large collection of node-sets, each a subset of an annotated community, and for each of these sets we perform the same roundness calculation as before. In this case, because many of these community parts were quite small, we had to consider smaller sets than in the previous experiment, and so we allow sets of any size. The fourth row of Table 4.1 contains the results for this experiment. We see that these ratios are much closer to 1 than were the ratios in the earlier experiment, indicating that these community parts are generally much "rounder" than the larger annotated communities.

Finally, we examine how these parts are related to one another. For every annotated community, we define a new network N in which every node $n_1, n_2, ..., n_k$ in N represents one of the parts $P_1, P_2, ..., P_k$ of that annotated community, obtained through the Louvain method. Because we only consider connected communities, for every part P_i, there is at least one edge between a node in P_i and a node in P_j, for some $j \neq i$. For each P_i, we calculate T_i, the total number of edges outgoing from nodes in P_i to nodes in other parts. Then, for each P_j, if there are at least $\frac{T_i}{k-1}$ edges from nodes in P_i to nodes in P_j, we connect nodes n_i and n_j in N. For example, if there are a total of 6 parts, and there are 100 edges outgoing from nodes in P_i to nodes in other parts, then we connect P_i and P_j if there are at least 20 edges from nodes in P_i to nodes in P_j. Note that every pair (P_i, P_j) is considered twice: once when we consider the total number of edges outgoing from P_i, and once when we consider the total number of edges outgoing from P_j. It may be, for instance, that if P_i is very small and P_j is very large, a large portion of P_i's outgoing edges go to P_j, but only a small portion of P_j's outgoing edges go to P_i. In this case, as long as the condition is met at least once, we connect the two nodes. Note that it is possible that this network of parts might be disconnected. In practice, for most of the network datasets we considered, this did not occur, and for those network datasets where it did happen, fewer than 1% of the annotated communities produced such a structure. When we did encounter such a situation, we were unable to calculate roundness, and so discarded such sets from consideration.

We then repeat the same experiment for these networks of parts. The results are contained in the fifth row of Table 4.1. From these values, it appears that these networks fit the "round" model very well. However, we caution that these networks are often very small: for all datasets, the vast majority of these networks have fewer than 10 nodes (that is, most annotated communities decomposed into fewer than 10 parts), and so have very small diameters. These ratios would have been more informative if these networks were larger; nevertheless, this is a reasonably good indication that the parts within annotated communities are well-connected to one another, as opposed to being positioned in a structure like a path graph.

We next supplement these results by calculating values of other network features in addition to diameter, and then repeating the above experiments. For each annotated community, community part, or network of parts, we calculate the following features, and compare them to the value of those features on a random graph:

- Median edge betweenness: To calculate edge betweenness, for every pair of nodes, we identify all shortest paths between those two nodes. The edge

Table 4.2: Ratios of median edge betweenness of annotated communities, annotated community parts, and networks of parts to median edge betweenness of random graphs of the same size.

	Grad	Ugrad	HS	SC	DM	Ama.	Manu
Ann. Comm. Ratios (≥ 50)	0.74	0.88	0.85	0.83	0.77	0.77	0.70
Comm. Parts Ratios	0.97	0.95	0.96	0.98	0.99	0.99	1.01
Comm. Parts Network Ratios	1.02	1.02	0.98	1.00	1.03	1.00	1.00

Caption: Edge betweenness values again demonstrate that the annotated communities do not resemble "round" graphs, but they are composed of "round" parts that are connected to one another in a "round" way.

betweenness of an edge is defined as the fraction of all such shortest paths that that edge appears in. Each edge has its own edge betweenness value, and for each graph (annotated community, community part, or network of parts), we identify the median edge betweenness value [8].

- Transitivity: The transitivity of a graph is defined as the fraction of all pairs of adjacent edges $(a, b), (b, c)$ for which nodes a and c are connected [65].

Tables 4.2 and 4.3 contains these ratios. Once again, we see that for every network, both the annotated community parts and networks of parts resemble random graphs much more closely than do the annotated communities.

The fact that parts of annotated communities closely resemble random graphs is valuable, but somewhat unsurprising. These parts were identified through use of greedy modularity optimization, which is based on the "round" model of communities, and so it is natural that the communities that it finds fit that model. The purpose of these experiments was not to demonstrate that

Table 4.3: Ratios of median transitivity of annotated communities, annotated community parts, and networks of parts to median transitivity of random graphs of the same size.

	Grad	Ugrad	HS	SC	DM	Ama.	Manu
Ann. Comm. Ratios (≥ 50)	4.99	2.20	6.35	4.65	2.59	9.25	3.09
Comm. Parts Ratios	1.23	1.26	1.92	1.62	1.20	1.47	1.07
Comm. Parts Network Ratios	1.08	0.95	1.11	1.06	1.03	1.09	1.0

Caption: Transitivity values again demonstrate that the annotated communities do not resemble "round" graphs, but they are composed of "round" parts that are connected to one another in a "round" way.

sets found by greedy modularity optimization are "round," but rather to argue that in general, annotated communities are not round, but for a given annotated community, there generally exists a partitioning of that community such that each individual part is "round" and that the parts are connected to one another in a "round" way. Working from this intution, we next present the Node Perception template for finding overlapping communities.

4.4 Node Perception Algorithm Template

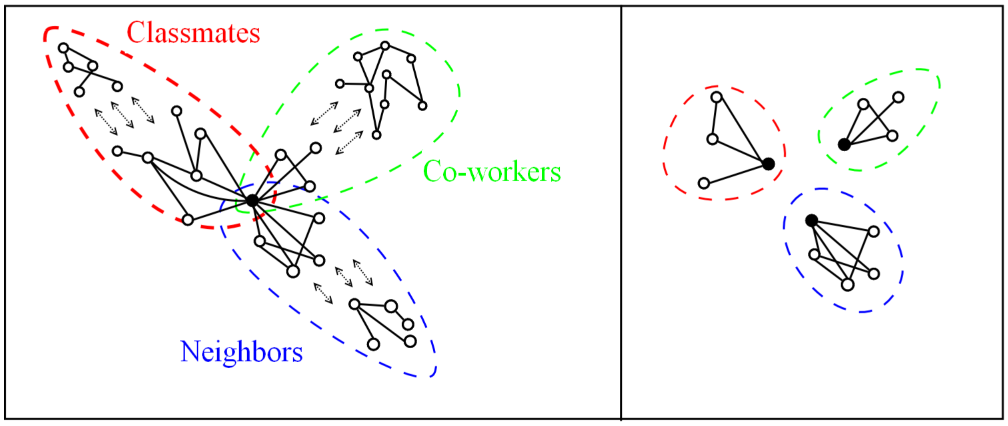
Using the results from the previous section, we now present our Node Perception algorithm template for finding overlapping communties. Unlike many other popular methods, we do not attempt to optimize one particular mathematical measure. Rather, we describe a flexible template that is based on the intuition that communities are made up of small groups that are linked together. Because the template is not based on any one measure, users are able to use features of a network, as well as running time and memory constraints, to create an algorithm appropriate for their needs. In this section, we give a broad overview of our template, and then discuss the various implementations that we analyze.

4.4.1 Overview

Our method consists of three parts. For a network G, for every node v in G, we separate v's neighbors into sub-communities, each containing a portion of a larger community to which v belongs. Next, we create a new network H, such that each node in H represents a sub-community from the first step. Two nodes in H are connected if their associated sub-communities in G are related in some way (e.g., share some number of elements). Finally, we identify communities in H. To identify communities in the original network G, we decompose each community in H into its member sub-communities, and each sub-community into its member nodes from G. Each node from G can appear in multiple sub-communities (nodes in H), and thus can appear in multiple communities. Many different methods for performing each step are possible. For a graphical illustration of this process, see Figures 4.1 and 4.2.

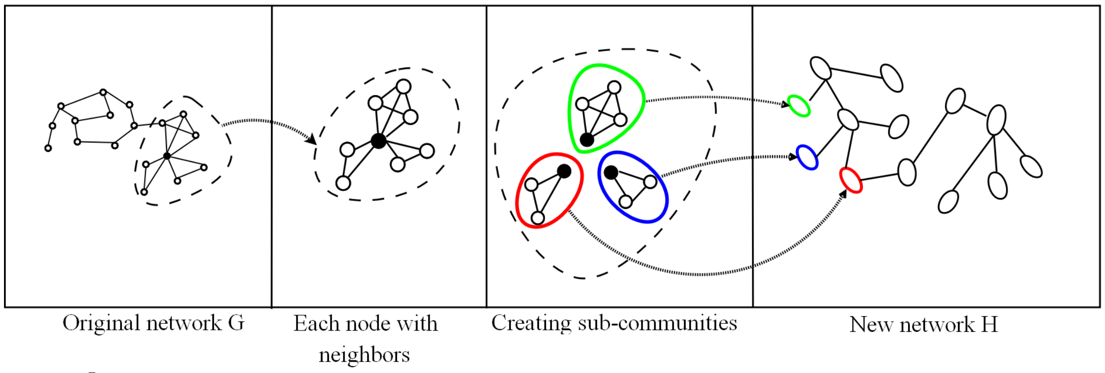
In the next section, we explain each of these three steps in greater detail. Because each step can be performed in many different ways, portions of the next section are intentionally non-specific. We present precise details of our implementations in Section 4.5.

Figure 4.1: Creation of sub-communities



Caption: Illustration of how one identify sub-communities.

Figure 4.2: Algorithm overview



Caption: Illustration of Node Perception process.

4.4.2 Detailed Description

Creation of Sub-Communities

Our analysis in Section 4.3 demonstrated that annotated communities may be well-described by a model that joins together small "sub-communities." In this paper, we create these sub-communities by considering the node neighborhoods of each individual node. Work by Gleich and Seshadhri, demonstrating that node neighborhoods typically have low conductance scores [21], provides some justification for this method, but other methods are likely possible.

The first step of this method is to identify sub-communities in network G. To do this, we iterate over each node v in network G and group v's neighbors into sub-communities. A node v may belong to many large communities $C_1, C_2, ...$, each containing v and some of v's neighbors, as well as more distant nodes that v is not connected to. Each sub-community $S_{1,v}, S_{2,v}...$ contains v and those of v's neighbors that belong to $C_1, C_2, ...$; i.e., the sub-communities correspond to v's local perception of the large communities. For example, in a social network, separate sub-communities may represent the groups of an individual's friends from college, friends from a soccer team, acquaintances from work, and so on. When we consider the subgraph immediately around a vertex v, those of v's neighbors that know one another from the same community are likely to be better connected to one another than to those of v's neighbors that v knows from a different group (e.g., v's friends from a soccer team likely know each other, but are less likely to know v's co-workers). It seems, then, that on this small subgraph containing only v's neighbors, a reasonable community detection algorithm should easily be able to decompose the nodes into clear, separate sub-communities.

The method used to create this grouping should be suitable for the features of G as well as the data available. If one believes that each of v's neighbors is likely to belong to only one of v's communities, one could apply a partitioning algorithm to the subgraph of G induced by v's neighbors. For greater accuracy, one could include not only v's immediate neighbors, but also the neighbors of those neighbors. One can also use a method for detecting overlapping communities. This method could be, for example, the Link Communities method, Clique Percolation, or even Node Perception used recursively. In this paper, we use simple partitioning methods, which are sufficient for good performance on many types of networks.

The same sub-community may be created multiple times: for example, when considering sub-communities centered around node a, we may create sub-community $\{a, b, c\}$. Later, when considering sub-communities around node b, we may again create sub-community $\{a, b, c\}$. We choose to allow multiple copies of the same sub-community, but one could also choose to consider only one copy.

This step was based on the intuitive notion that although each person belongs to many communities, relationships between individuals can typically be neatly placed into one community. However, the success of this step does not depend on this intuition. Consider a pair of siblings A and B from some family. Their relationship may fall into many communities, such as the family itself, a local school, a neighborhood sports team, etc. When using a partitioning algorithm to find sub-communities centering around A, we will place B into only one of A's sub-communities (e.g., the one corresponding to family). However, even in this case, other of B's neighbors from her other communities may place B into sub-communties representing those other communities: although A does not primarily think of B as "a person with whom I play sports", someone else likely does. Thus, even when some relationships fall into many different communities, each node might still be placed into at least one sub-community for each of its communities. While there are obvious theoretical counterexamples to these assumptions, our results provide strong evidence that they are generally valid.

A unique and particularly useful feature of our approach is that supplemental data about the network can be used to improve this initial grouping of nodes into sub-communities. For example, if G is the Facebook network, then in addition to using a partitioning method to create the sub-communities, groups of people frequently appearing together in Facebook photographs could be used to create additional sub-communities. If G is a co-authorship network, then an additional sub-community could be a group of people who wrote a paper together. In contrast, with an algorithm centered about some mathematical optimization, it is unclear how to use such information. If a set of people within a community appears in pictures together, that does not modify, say, the conductance of that community. Because our evaluation methods rely on metadata, we are unable to test such data inclusion methods in this paper (with the exception of a case study in Section 4.8.3); however, while our results demonstrate that grouping based solely on the network structure is sufficient for this method to exceed the performance of other community detection algorithms, we fully expect supplemental data to boost its performance further.

Creation of Sub-Community Network

In the second step, we create a new network H such that every node in H represents one sub-community from the first step. Two nodes in H are joined by an edge if the sub-communities that they represent are related (e.g., if they share any nodes in G). This edge may be weighted in relation to the strength of the relationship between the two sub-communities. One may also require that two sub-communities share some threshold number of nodes before connecting them in H: this will result in a sparser network H, and so will speed up the processing in the third step.

Identifying Communities in the Sub-Community Network

In the third step, based on the results from Section 4.3 that showed that small portions of a community are typically well-connected to one another, we use an existing community detection algorithm to group the elements of H into communities. As with the first step, a method appropriate to the particular application should be used. If speed is important, a fast partitioning method such as greedy modularity optimization may be appropriate, or one could even set a minimum edge weight and then find connected components. To find a specific number of communities, a clustering algorithm may be best. If a slower method is acceptable, then, as in step 1, a method for finding overlapping communities can be used.

To determine the communities in G, for each community C in H, simply take the union of all nodes from G that are contained in the sub-communities represented by the nodes in C. Because each node v can appear in multiple sub-communities in step 1, and thus may be represented by multiple nodes in H, v may appear in multiple communities in G. One could require that a node v from G appear in at least k sub-communities from C, for some k, (i.e., at least k nodes from G have "voted" to place v in community C), and this results in smaller, more tightly-knit communities, with the possibility that some nodes from G will not appear in any communities. Alternatively, the number of times a node v appears in a sub-community in C could be interpreted as related to the strength of v's membership in C.

4.5 Specific Details of our Implementations

We consider various specific implementations of the Node Perception template, and present a few representative examples here. Naturally, not every implementation is appropriate for every network; in particular, some of the community detection algorithms that we use in our templates are computationally intensive and unsuitable for large or especially dense networks.

Our implementations are distinguished from other similar algorithms in two important ways. First, other algorithms typically have rigid conceptions of "sub-communities." For example, in Link Communities, "sub-communities" are simply edges from the original network, and Clique Percolation requires sub-communities to be cliques of a fixed size. In contrast, our methods use a community detection method to find sub-communities. This means that sub-communities may have varying sizes and degrees of connectivity, depending on the structure of the network in that location. Second, other methods typically join sub-communities together by identifying connected components. Node

Perception, however, can require more than simple connectedness: using a community detection algorithm to group the sub-communities ensures a high degree of connectivity between these local groups.

In each implementation described, we perform the first step of identifying sub-communities as follows: for every node v in network G, we consider the subgraph K_v induced by v's neighbors (but not v). We then apply various community detection methods to K_v to partition v's neighbors into disjoint sets (described in greater detail below). To each of these sets, we add v, and thus obtain the sub-communities.

In network H, we create one node representing each sub-community from the first step. If the same sub-community is created multiple times in the first step, we allow it to appear multiple times in network H. In order for two nodes in H to be connected, we require that they have a minimum Jaccard similarity with each other. We chose to require a minimum Jaccard similarity rather than a minimum number of shared nodes because we wanted the amount of required overlap to be greater for larger sub-communities[4]. For this implementation, we require a Jaccard similarity of at least 0.2. In addition to setting this threshold, we weight each edge in H by the Jaccard similarity of the two adjacent sub-communities.

We also attempted to use methods for finding overlapping sub-communities, but this resulted in a large increase in the number of sub-communities, sometimes making network H impractically large. We also experimented with other methods of creating network H, such as setting different similarity thresholds, but the experimental results were effectively the same, and so we do not present

[4]The Jaccard similarity of two sets is defined as the number of elements contained in the intersection of those sets divided by the number of elements in their union [28].

them here.

Each implementation that we describe produces a set of overlapping communities. Two communities in this set may be very similar or even identical. As a final clean-up procedure, we remove duplicates, but allow similar sets to remain. For some networks, the number of communities output may thus be quite large.

Modularity-Modularity (Mod-Mod): In the Mod-Mod implementation, we create the initial sub-communities by applying Louvain greedy modularity optimization to each K_v subgraph. Network H is formed as described above, and we use Louvain greedy modularity optimization to partition the nodes of network H.

Of all the implementations that we consider, Mod-Mod and IM-Mod (described next) are based most strongly on the intuitions that we gained in Section 4.3. Modularity maximization algorithms attempt to find "round" communities, with connections throughout the entire set. As we saw in Section 4.3, when we partition an annotated community into smaller communities, those smaller communities tend to be "round." Similarly, when we analyze the connections between those smaller communities, we see that they tend to be connected in a "round" manner. In this implementation, we thus use modularity maximization both to identify sub-communities as well as to join those sub-communities.

Pseudocode for this process is presented in Algorithm 1.

Infomap-Modularity (IM-Mod): IM-Mod is identical to Mod-Mod, except we use the Infomap partitioning algorithm to identify sub-communities. As with Mod-Mod, this method finds "round" sub-communities, and joins them together in a "round" way.

Components-Modularity (Comps-Mod): The Comps-Mod implementation is the same as Mod-Mod and IM-Mod, except that the sub-communities are formed by identifying connected components of each node neighborhood K_v. Although a connected component is not necessarily "round," this method is likely to be faster than either of the previous two methods.

Modularity-Link Communities (Mod-LC): In previous implementations, we used greedy modularity optimization to identify communities in H. However, this method is known to suffer from certain flaws; namely, the existence of modularity's "resolution limit," which hinders its ability to find communities in very large networks [18]. Because the networks H of sub-communities may be quite large, we also consider other community detection methods.

In the Mod-LC implementation, we again use greedy modularity optimization to identify sub-communities, create network H as described earlier, and then use the Link Communities method to identify communities in H.

Because Link Communities is a very memory intensive method, we were unable to apply this implementation to one especially dense network (network DM).

Modularity-Node Perception (Mod-NP): In the Mod-NP implementation, we again create network H using greedy modularity optimization to find sub-communities, and create network H as described before. We then use the Node Perception Mod-Mod algorithm to find communities in H. Note that because network H is typically larger than the original network G, Node Perception cannot properly be considered a recursive algorithm, as there is no clear stopping

ALGORITHM 1: Mod-Mod pseudocode

```
Input: A network $G$ with vertices $V(G)$ and edges $E(G)$
Output: Communities in network $G$
Multiset $SUBCOMMS = \emptyset$; Set $ALLCOMMS = \emptyset$;
for every $v$ in $V(G)$ do
    $K(v) = \{x : (x, v) \in E(G)\}$ ; $L$ = Louvain($K_v$); for every set $S$ in $L$ do
        $S = S \cup \{v\}$; Add $S$ to $SUBCOMMS$;
    end
    Create network $H$, where $|V(H)| = |SUBCOMMS|$; Define mapping
    $f : V(H) \rightarrow SUBCOMMS$ for every $s \in V(H)$ do
        for every $t \in V(H)$ do
            if $JaccardSimilarity(f(s), f(t)) \geq 0.2$ then
                Add $(s, t)$ to $E(H)$;
            end
        end
    end
    $P$ = Louvain($H$); for every $P_i \in P$ do
        $C = \emptyset$; for every $s$ in $P_i$ do
            $C = C \cup f(s)$;
        end
        Add $C$ to $ALLCOMMS$;
    end
    Return $ALLCOMMS$;
end
```

condition. However, subject to memory and running time constraints, one can repeatedly apply Node Perception an arbitrary number of times. While for some smaller networks, we were able to perform many steps of recursion, only 1 or 2 steps were possible for most networks. Indeed, as with Mod-LC, we were again unable to apply this method with even 1 step of recursion to one network (DM).

For the other networks, the results presented were obtained by using 1 step of recursion.

DEMON, or Label Propagation-Subset (LP-Subs): The DEMON (Democratic Estimate of the Modular Organization of a Network) algorithm is a specific instance of our Node Perception template, and was created independently by Coscia, et al. [12]. In this algorithm, the initial identification of sub-communities is performed using the Label Propagation algorithm described in [51]. The algorithm contains a tunable parameter that controls the identification of communities in network H; however, in the version that is analyzed most completely in [12] and that we consider here, the communities in H consist of those sub-communities C such that there is no other sub-community D that is a proper superset of C. Observe that in this method, communities are limited to a maximum diameter of 2, because every community is also a sub-community, and every sub-community is contained within the immediate neighborhood of some central node. Because the Label Propagation and subset determination steps are quite fast, this method scales very well.

4.6 Evaluation Overview

In this section, we evaluate the implementations discussed in the last section, as well as several other popular community detection methods.

Often, researchers in this area determine the quality of a community detection algorithm by evaluating its output with respect to a formal mathematical conception of what a community ought to look like. This approach has advantages, but it is unclear that any particular mathematical definition is correct. Another common approach is to use synthetic benchmark networks with planted communities, but this method is susceptible to similar problems. Because, despite years of research into social networks, there is still little consensus on what a "real" community is, we choose to take a different approach to evaluating community detection algorithms. Rather than assuming that a particular mathematical definition is correct, we evaluate algorithms through use of the annotated communities described in Chapter 2.

This metadata describes characteristics of each node, and can be used to identify sets of similar nodes. In some cases, the metadata identifies sets of nodes that intuitively seem to be good communities: for example, graduate students in a particular department probably form a good community. In other cases, the metadata might identify sets of nodes that are seemingly poor communities. This is not a problem with the approach that we take in this paper. Because our goal is to compare our algorithms' performances against those of other algorithms, we do not compare against an absolute standard: if a particular annotated community is not represented in the network structure, then we can expect that none of the algorithms will recover it, so each algorithm will be penalized equally for failing to recover it.

Our evaluation approach differs from the norm, but we note that similar strategies have been successfully used by other researchers, such as Ahn, et al. (who incorporated metadata information into their evaluation of the Link Communities algorithm) [4] and Backstrom, et al. (who used metadata to study the evolution of communities over time) [6]. Although other evaluation methods are common, they are based on *a priori* assumptions about the structure of communities and networks, and so can be biased towards particular algorithms. In contrast, using metadata to identify annotated communities allows for an equal, unbiased comparison between many different types of algorithms.

4.7 Methods and Results

For each network, we run our Node Perception templates Comps-Mod, IM-Mod, Mod-Mod, Mod-LC, Mod-NP, and LP-Subs, as well as Infomap (IM), Louvain method for greedy modularity optimization (Mod), Link Communities (LC), Clique Percolation (CP), and OSLOM (OS). Our results show that the various Node Perception methods generally outperform all other community detection methods.

For Mod-LC, due to running-time concerns, we used an approximation to the Link Communities algorithm for networks Amazon and DM. Rather than finding the threshold link weight that allows network H to be split at the optimal partition density, we consider thresholds in $0.1, 0.2, ..., 0.9$, compute the partition densities obtained by each partitioning, and select the partitioning that obtains the greatest partition density. In experiments on smaller networks, the results of this approximation method were largely indistinguishable from the original method.

The Infomap algorithm is non-deterministic, and one parameter of the algorithm specifies the number of partition attempts. For the smaller networks, we used 10 iterations, but for the larger Amazon and DM, we used only 1 iteration.

Clique Percolation requires the user to specify the clique size. On these networks, a size of $k = 4$ gave the best overall results, and so we present those results here. Networks Undergrad, DM, and SC had especially dense regions, and the Clique Percolation algorithm required an infeasibly large amount of memory and running time, so we do not present its results for these networks (and thus, also do not present its overall results).

OSLOM produces several output files, each corresponding to a different level in the hierarchy of communities. For these networks, using the results for the lowest level of the hierarchy produces the highest accuracies, and so we present those results.

4.7.1 Evaluation Methods

Our evaluation strategy relies on the use of the Jaccard similarity index to determine when an algorithm has recovered an annotated community.

For each network N and algorithm, we compare each annotated community A in N to the communities C found by that algorithm. For each annotated community A, we calculate the maximum Jaccard similarity J_A between A and a community C found by the algorithm, over all communities found by the algorithm (that is, J_A is the Jaccard similarity between A and the "closest" detected community). We then take the average of all such J_As and report this value in Table 4.4. This is the "Continuous Recall" score of the algorithm. Although other methods of comparison are certainly possible, we chose to use Jaccard similarity because it takes into account the size difference between the two sets being compared. If, for example, a community detection algorithm returns the entire network as one community, then it certainly contains every annotated community, but it has given us no useful information.

We additionally calculate a binary version of recall, which sets a threshold for Jaccard similarity. In this measure, we say that an algorithm "found" an annotated community if it recovered it with a Jaccard similarity of at least $\frac{1}{3}$. A Jaccard similarity of $\frac{1}{3}$ indicates that if the detected community is exactly the same size as the annotated community, as few as half the elements from the annotated community were found, but if more elements are found, allows a size differential of up to a factor of 3. We experimented with other thresholds, and the algorithms each scored similarly relative to one another. Binary recall is similar to continuous recall, but gives a better sense of the distribution of Jaccard similarities: e.g., if an algorithm has a moderately high continuous recall score, binary recall helps us understand whether this is because it found a few communities very well, or many communities moderately well. Table 4.5 reports the binary recall scores for each network and algorithm.

We also compute the "Continuous Precision" and "Binary Precision" scores of the algorithm by performing the opposite calculation: for each community C found by the algorithm, we compute the maximum Jaccard similarity J_C between C and an annotated community A, and report the average of all such J_Cs (Table 4.6), or the fraction of such J_Cs that are at least $\frac{1}{3}$ (Table 4.7).

We caution that recall and precision should be interpreted carefully when comparing a partitioning algorithm to an algorithm that finds overlapping communities. A partitioning method, when compared to a method for finding overlapping communities, will typically find a smaller number of strong communities, and so may have lower recall and higher precision scores (indeed, our results show that the algorithm with the lowest recall has the highest precision).

In all tables, the best performing algorithm for each network is presented in boldface. Additionally, because we are interested in comparing methods for finding overlapping communities, the best performing overlapping community detection method is italicized in Tables 4.6 and 4.7.

4.7.2 Results

Our results show that across 7 datasets, the Node Perception algorithms have significantly higher continuous recall scores than the other methods (Table 4.4). Of the algorithms that finished on every network, on average, every Node Perception method outperformed every non-Node Perception method. IM-Mod and Mod-Mod had an average recall of 0.38, a 15% improvement over Link Communities, the best non-Node Perception algorithm. Comps-Mod and LP-Subs (DEMON) did not perform as well as the other Node Perception methods, but on average, still outperformed every non-Node Perception algorithm. We see qualitatively similar results when evaluating algorithms using binary recall (Table 4.5). The output from Mod-LC performed comparably with IM-Mod and Mod-Mod, but was much slower because the Link Communities method is slower than greedy modularity optimization. The Mod-NP implementation produced dramatically better recall scores, but could not be applied to network DM. On 5 of the 6 networks on which we ran Mod-NP, it outperformed every other algorithm, often by a large margin.

As expected, the two partitioning methods had the highest precision scores (Tables 4.6 and 4.7). Of the Node Perception implementations, Mod-Mod had the worst continuous precision, but it was still slightly better than the best of the other overlapping community methods (Link Communities and OSLOM).

Table 4.4: Recovery of annotated communities (continuous recall)

	Amaz.	Grad	Ugrad	HS	SC	DM	Manu	Avg.
NP: Comps-Mod	0.47	0.56	0.27	0.19	0.21	0.25	0.58	0.36
NP: IM-Mod	0.41	0.57	0.32	0.20	0.23	0.36	0.59	0.38
NP: Mod-Mod	0.45	0.54	0.31	0.19	**0.25**	**0.38**	0.56	**0.38**
NP: Mod-LC	0.42	0.54	0.34	0.19	0.23	–	0.56	–
NP: Mod-NP	**0.50**	**0.61**	**0.35**	**0.21**	0.24	–	**0.61**	–
NP: LP-Subs	0.42	0.53	0.29	0.19	0.22	0.29	0.54	0.35
LC	0.41	0.50	0.19	0.17	0.20	0.33	0.55	0.33
CP	0.36	0.47	–	0.11	–	–	0.31	–
OSLOM	0.43	0.58	0.17	0.14	0.16	0.22	0.55	0.32
IM	0.44	0.60	0.24	0.12	0.15	0.16	0.57	0.33
Mod	0.13	0.42	0.24	0.03	0.05	0.05	0.57	0.23

Caption: For each annotated community A, find the detected community C such that the Jaccard similarity J_A of A and C is maximized. This table presents the average of the J_A values over all annotated communities A in a network.

The other Node Perception algorithms performed much better. We again see that Mod-NP performs very well. On 3 of the 6 networks on which Mod-NP was evaluated, it had the highest binary precision score of all methods for finding overlapping communities (and in one case, it even outperformed the two partitioning methods). In most cases, it outperformed the non-Node Perception algorithms by a very large margin (e.g., on network Ugrad, it has a binary precision of over 0.5, while LC and OSLOM are both below 0.1). Mod-LC and

Table 4.5: Recovery of annotated communities (binary recall)

	Amaz.	Grad	Ugrad	HS	SC	DM	Manu	Avg.
NP: Comps-Mod	0.67	**0.71**	0.34	0.13	0.14	0.25	0.70	0.42
NP: IM-Mod	0.53	**0.71**	0.39	0.16	0.19	0.46	0.70	0.45
NP: Mod-Mod	0.63	0.67	0.39	0.16	**0.26**	**0.50**	0.70	**0.47**
NP: Mod-LC	0.62	**0.71**	0.39	0.17	0.25	–	0.70	–
NP: Mod-NP	**0.74**	**0.71**	**0.41**	**0.19**	0.23	–	**0.80**	–
NP: LP-Subs	0.61	**0.71**	0.37	0.11	0.14	0.43	0.70	0.43
LC	0.54	0.63	0.22	0.09	0.19	0.34	0.60	0.39
CP	0.44	0.54	–	0.00	–	–	0.30	–
OSLOM	0.58	**0.71**	0.24	0.09	0.12	0.18	0.60	0.36
IM	0.58	**0.71**	0.22	0.04	0.05	0.16	0.60	0.34
Mod	0.15	0.50	0.22	0.00	0.00	0.05	0.60	0.22

Caption: For each annotated community A, find the detected community C such that the Jaccard similarity J_A of A and C is maximized. This table presents the fraction of annotated communities A such that $J_A \geq \frac{1}{3}$.

LP-Subs (DEMON) also performed very well.

Although Mod-NP appears to be the best implementation of Node Perception (subject to efficiency constraints), the much simpler implementations often perform nearly as well. Consider the binary recall for network Ugrad: all six Node Perception implementations score between 0.34 and 0.41, while the next best algorithm scores a 0.24. We see similar behavior on network HS.

Table 4.6: Recovery of annotated communities (continuous precision)

	Amaz.	Grad	Ugrad	HS	SC	DM	Manu	Avg.
NP: Comps-Mod	0.12	0.25	0.22	0.05	0.07	0.04	0.66	*0.20*
NP: IM-Mod	0.13	0.22	0.14	0.05	0.06	0.04	0.63	0.18
NP: Mod-Mod	0.10	0.15	0.12	0.05	0.06	0.04	0.42	0.13
NP: Mod-LC	*0.14*	0.20	0.20	***0.07***	0.07	–	0.40	–
NP: Mod-NP	0.14	*0.28*	*0.33*	0.05	0.07	–	*0.68*	–
NP: LP-Subs	0.14	0.24	0.21	0.06	*0.08*	*0.05*	0.65	0.20
LC	0.09	0.13	0.06	0.04	0.06	0.03	0.48	0.13
CP	0.08	0.09	–	0.04	–	–	0.25	–
OSLOM	0.06	0.11	0.10	0.01	0.03	0.01	0.60	0.13
IM	0.17	0.37	0.60	0.03	0.05	0.10	**1.00**	0.33
Mod	**0.18**	**0.51**	**0.81**	0.03	**0.12**	**0.12**	**1.00**	**0.40**

Caption: For each detected community C, find the annotated community A such that the Jaccard similarity J_C of C and A is maximized. This table presents the average of the J_C values over all detected communities C in a network.

4.7.3 Discussion

The main implication of these results is that the general principle of joining together small sub-communities of varying size and structure is of far more importance than the specific implementation used. These results confirm the analysis in Section 4.3, which showed that annotated communities are formed of well-connected sets that are well-connected to one another. While the resource-

Table 4.7: Recovery of annotated communities (binary precision)

	Amaz.	Grad	Ugrad	HS	SC	DM	Manu	Avg.
NP: Comps-Mod	0.104	0.286	0.121	0.005	0.007	0.012	***1.0***	*0.219*
NP: IM-Mod	*0.125*	0.241	0.101	0.004	0.005	0.019	0.878	0.196
NP: Mod-Mod	0.062	0.076	0.032	0.003	0.006	0.019	0.657	0.122
NP: Mod-LC	0.108	0.149	0.165	0.004	0.004	–	0.394	–
NP: Mod-NP	0.109	*0.376*	*0.506*	***0.006***	0.008	–	0.920	–
NP: LP-Subs	0.124	0.299	0.128	0.006	*0.010*	*0.019*	0.930	0.217
LC	0.044	0.064	0.004	0.001	0.005	0.007	0.764	0.127
CP	0.0.020	0.031	–	0.000	–	–	0.250	–
OSLOM	0.042	0.104	0.084	0.001	0.005	0.003	0.500	0.106
IM	0.194	0.419	0.643	0.006	**0.010**	0.115	**1.000**	0.341
Mod	**0.198**	**0.727**	**0.900**	0.000	0.000	**0.189**	**1.000**	**0.430**

Caption: For each detected community C, find the annotated community A such that the Jaccard similarity J_C of C and A is maximized. This table presents the fraction of detected communities C such that $J_C \geq \frac{1}{3}$.

intensive Mod-NP method gives dramatically better results than any other method that we analyzed here, even simple, fast methods such as Comps-Mod are generally much better than any of the other methods for finding communities. Thus, rather than considering various instances of this template in isolation, we ought to credit the success of these methods to the general principles of the Node Perception template. This view allows a practitioner to modify the template in order to suit his or her needs and to suit the features of the network under consideration.

4.8 Scalability and Suitability

Next, we discuss ways to modify the Node Perception algorithm to make it suitable for different types of networks.

4.8.1 Running Time

With the exception of network DM (discussed later), the Node Perception implementations finished quickly on all networks using a desktop computer. On Amazon, our largest dataset, Mod-Mod took approximately 15 minutes, and other networks took between a few seconds and a few minutes. Mod-Mod's running time was comparable to other methods for finding overlapping communities. Mod-LC and Mod-NP generally took only slightly longer (again, except for DM).

The slowest step in our algorithm is locating communities in the network H of sub-communities. In general, the number of nodes in H will be $m\,|V(G)|$, where m is the average number of communities that each vertex belongs to. The number of edges in network H is more difficult to analyze, and, as with Clique Percolation, depends on the structure of dense regions of G. Network DM possesses areas with a large amount of sub-community overlap, and thus was very slow for Clique Percolation, Node Perception, and Link Communities: on a cluster node with 16GB of memory, Mod-Mod took approximately 12 hours to terminate and Link Communities took approximately one day. Clique

Percolation, Mod-LC, and Mod-NP would have taken much longer (weeks or months) if we had allowed them to terminate.

Although Mod-LC and Mod-NP were inappropriate for network DM, this should not be interpreted as a flaw of Node Perception itself: due to its template nature, one can produce both efficient and inefficient implementations. In cases such as network DM, when network H is very dense, H's size can be reduced by filtering out smaller sub-communities, requiring a greater amount of overlap between sub-communities, or using a simple clustering scheme to identify communities in H, rather than a slower, more resource intensive method such as Link Communities.

4.8.2 Network Suitability

The Node Perception algorithm as we have presented it is highly dependent on network transitivity. To create the sub-communities, we consider the neighbors of a node and their connections to one another. Creating sub-communities thus relies heavily on these neighbors being connected to other neighbors in the same large community. Generally, many networks do possess high transitivity, which is one reason why our approach was successful.

However, there may be networks with low transitivity that still possess community structure. For instance, instead of 3-cycles (transitivity) the network may instead have many 4-cycles. In such a case, we might modify the method used to identify the sub-communities. For example, if the network has many 4-cycles, then we might consider not just the neighbors of a node, but also those neighbors' neighbors.

We emphasize that such modifications are generally unnecessary, and the unmodified version of the algorithm worked well on all datasets evaluated. Moreover, one can ascertain the need for modification by examining the network structure by sampling random nodes and progressively widening the diameter of their “local” neighborhoods until connections between nodes in the neighborhoods are found. One might even select different diameters for different nodes, depending on the graph structure around those nodes.

4.8.3 Implementation Selection and Case Studies

Our results have shown that many implementations of Node Perception will outperform other algorithms. In this section, we provide guidance and examples for users wishing to choose between various implementations.

First, observe that the simple Mod-Mod implementation works efficiently and accurately for all networks examined. It gives higher overall continuous and binary recall scores than all non-Node Perception algorithms that we considered, and gives a higher continuous precision score than the other algorithms for finding overlapping communities (although Link Communities has a slightly higher binary precision score). Because the Louvain method for greedy modularity optimization is more memory efficient than many other methods, the Mod-Mod implementation runs easily for even large networks. However, Mod-NP tended to give higher binary and continuous recall scores than Mod-Mod, so it may be a better choice if the network is sufficiently small.

A user can also customize an implementation to the network being considered. To guide a practitioner in selecting an appropriate implementation of

Node Perception, we provide the following three case studies.

Network Grad

Network Grad possesses some important features: it is small, with only 503 nodes and 3256 edges, and as a Facebook network, is an incomplete representation of the true underlying social network. Due in part to its small size, the average degree of each node is less than 13, and so, when finding sub-communities, most subgraphs that we consider are quite small. Even if we consider the subgraphs obtained by going two steps out from each node, we still have a low average size of 76 nodes (in contrast, for network DM, each such subgraph would have an average size of approximately 1000 nodes). Thus, even if we consider these larger subgraphs, the Mod-Mod implementation is still likely to be fast. Moreover, because we know that the data is an incomplete representation of the complete network, considering these larger subgraphs will likely give us more accurate sub-communities, because the modularity algorithm can take advantage of information about nodes that were not included in the smaller subgraph. Indeed, applying this method to network Grad gives us a continuous recall score of 0.65 (as compared with 0.54 in the original Mod-Mod implementation), a binary recall score of 0.79 (instead of 0.67), a continuous precision score of 0.25 (instead of 0.15), and a binary precision score of 0.276 (compared with 0.076). We see that considering these larger subgraphs gives a startling increase in accuracy. Note, however, that going out too far from each central node will defeat the point of the algorithm, as the sub-communities are supposed to represent each node's perception of the network around it. The success of this method thus depended on two important features of the network: its small size (which allowed for this method to be reasonably fast) and its known incompleteness.

If one had a network with both sparse and dense sections, one could use a modification of this implementation by using the smaller subgraphs (i.e., those obtained by going one step out from the central node) in dense regions, but using the larger subgraphs in sparse regions.

Network SC

We now attempt to improve Node Perception's running time on network SC, while still obtaining good results. Although all of the Node Perception implementations that we considered previously were reasonably fast for network SC, we consider the following modifications as an example for how one might implement Node Perception for a network that is much larger.

As with many networks, SC has a degree distribution in which most nodes have a very low degree, but some nodes have a much higher degree [11]. The creation of sub-communities is thus usually fast, but sometimes slow. We demonstrated earlier that if we created sub-communities even through a method as simple as finding connected components, Node Perception produced reasonably strong results. Thus, to improve running time, we use greedy modularity optimization to partition the neighborhoods of low degree nodes, and use the faster method of finding connected components to partition the neighborhoods of high degree nodes (in this case, we define 'high degree' to be those nodes that are in the top 1% of nodes, as ranked by degree). We then again join the sub-communities using greedy modularity optimization. This process produces results that are better than Comps-Mod, and nearly as good as the original Mod-Mod algorithm, as measured by all four accuracy metrics (in all cases, the decrease in accuracy is measurable only in the third significant digit or later). The running time, however, is much improved, and as expected, lies between the running times for Mod-Mod and Comps-Mod.

Network HS

Network HS is a genetic interaction network, and in such networks, the functions of each gene, which we used to identify communities, are typically determined experimentally by biologists on a per-gene basis. Identifying these features can often be very time- and resource-intensive, and so it is highly likely that a practitioner will have incomplete community membership information [60, 5]. In this example, we demonstrate how one can use existing community membership information to boost the performance of Node Perception.

We use the standard Mod-Mod implementation; however, in addition to using the sub-communities generated by greedy modularity optimization, for each node n and every community C (genetic function) that n is known to belong to, we create an additional sub-community containing n and all of its neighbors from C. If n is known to play multiple genetic functions, then there may be multiple such sub-communities, and we add all of them to the list of sub-communities.

To evaluate this method, we use some of the annotated community data. For each node n in HS, we are given a set of known genetic functions that n plays. We had originally used these functions to identify the annotated communities; now, we randomly select half of these functions, and use this information to produce the additional sub-communities. This boosts the continuous recall of the Mod-Mod implementation from 0.19 to 0.25, the binary recall from 0.16 to 0.30, the continuous precision from 0.050 to 0.055, and the binary precision from 0.002 to 0.005. When applying this same methodology to network SC, which is also a genetic network, we see similar improvements in accuracy, with continuous recall increasing from 0.25 to 0.30, binary recall from 0.26 to 0.38, continous precision from 0.065 to 0.069, and binary precision from 0.006 to 0.009.

Naturally, this implementation does not allow for a fair comparison to other algorithms, because we are taking advantage of information not given to the other methods. However, the purpose of this case study is to illustrate the flexibility of the Node Perception template, and show how information, both internal and external to the network, can be used to increase its performance.

Any community detection methods can be used in the Node Perception template, and it is thus impossible to exactly characterize which methods are most appropriate for particular networks. We emphasize that basic, fast implementations are likely to work very well for most networks, but the template nature of Node Perception can also give users flexibility when desired.

4.9 Conclusion and Future Work

Mathematical formulations of community structure have traditionally fallen into one of two categories: those that are based on the belief that communities are "round" and well-connected throughout (such as a $G(n, p)$ graph), and those that are based on the belief that communities consist of small, tightly-connected groups that are well-connected to one another, although individual nodes themselves need not be well-connected to the rest of the community.

To carefully examine these general beliefs, we collected a set of seven network datasets of different scales and from different domains. Each of these networks contains some sort of external annotation that allowed us to identify "annotated communities." For example, in the product co-purchasing network Amazon, products were annotated with various categories (such as books by some author), and we grouped together all items from the same category into one community. For each of these annotated communities, we next produced a random graph of the same size, and calculated the ratio of the diameter of the annotated community to the diameter of the random graph. For every network, the annotated communities tended to have larger (in some cases, much larger) diameters, indicating that the "round" model of community may not be appropriate for these communities. We next used greedy modularity optimization to decompose each annotated community into several parts, and showed that these parts tended to be "rounder," with diameters closer to those of the random graphs. Finally, we examined how these parts are connected to one another, and showed that the graph representing these connections also fit the "round" model well. These results held even when we studied these graphs with features other than diameter, and suggest that communities are indeed made up of small, well-connected groups.

Using this intution, we developed the Node Perception algorithm template for finding overlapping communities in networks. In this template, we first identify sub-communities corresponding to each node's perception of the network around it. To perform this step, we consider each node individually, and partition that node's neighbors into communities using some existing community detection method. Next, we create a new network in which every node corresponds to a sub-community, and two nodes are linked if their associated sub-communities overlap by at least some threshold amount. Finally, we identify communities in this new network, and for every such community, merge together the associated sub-communities to identify communities in the original network.

We applied several different implementations of this method to each of our seven networks, and showed that, typically, all Node Perception implementations outperformed other considered community detection algorithms. It is particularly noteworthy that these results seemed to be independent of the specific implementation; this strongly suggests that the template itself, rather than a specific implementation, is responsible for this success. This conclusion is novel and valuable for two reasons: first, it allows a practitioner a great deal of flexibility in selecting an appropriate implementation, allowing him or her to take advantage of network features both internal and external, and second, our analysis into the structure of annotated communities, which was validated by the success of the Node Perception template, gives researchers insight into the fundamental nature of communities, justifying the general model of communities as collections of small, well-connected groups.

For future work, we are primarily interested in identifying formal methods for tailoring the Node Perception template to specific networks. In our case studies, we considered modifications based on both external knowledge about the network (such as its completeness or known community information) and knowledge gained from the structure of the network itself (degree distribution). To some extent, such modifications are necessarily ad hoc, as the range of possible external knowledge is far too wide to be captured with a finite set of rules. However, we may be able to formalize some modifications that are based on structural features of the network. Features considered may include transitivity, clustering coefficient [53], degree distribution, assortativity [47], and so on. Although even simple implementations of Node Perception were quite successful, such modifications may further improve performance.

Additionally, we are interested in creating successful community detection methods that are based on the same general principle of joining together small groups of nodes, but which do not necessarily identify these groups by examining node neighborhoods. When we analyzed the structure of communities, we decomposed each annotated community into several small groups by using greedy modularity optimization. How might an algorithm identify these groups, which while small, may not correspond exactly to a 'sub-community' as we have described in this paper? We are also interested in identifying other methods for joining together the small node-sets. It is possible that for other datasets, these node-sets are not joined together in a "round" way, and so different methods for connecting them may be useful.

CHAPTER 5

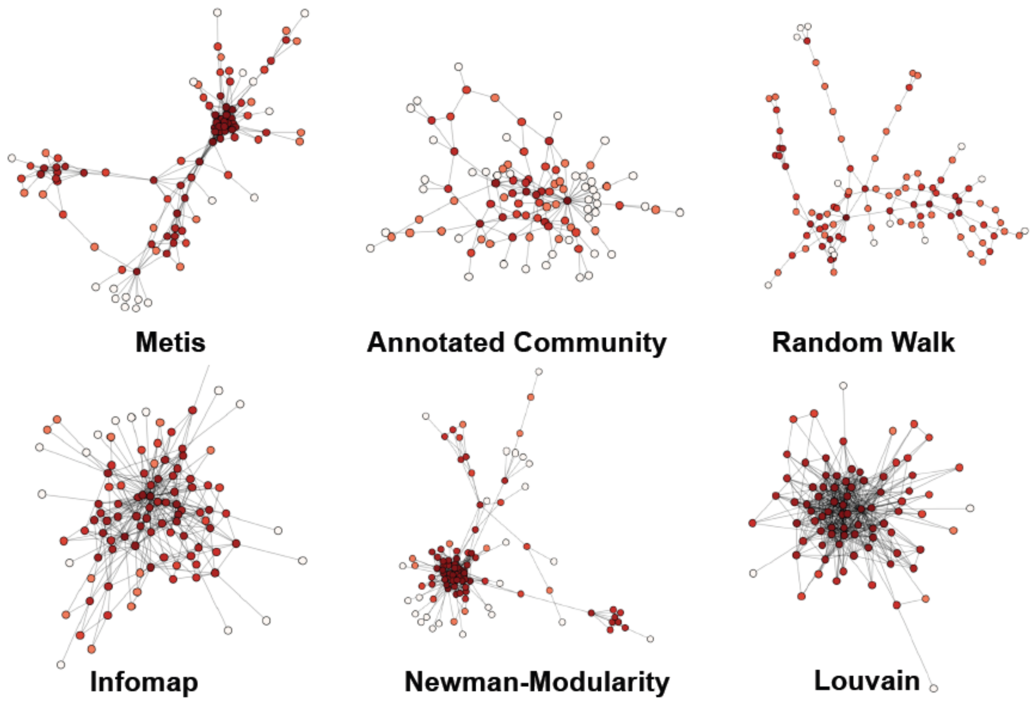
A MACHINE-LEARNING FRAMEWORK FOR UNDERSTANDING COMMUNITY STRUCTURE

We now present the first of two closely-related chapters in which we introduce and apply a machine-learning-based framework for comparing, contrasting, and analyzing communities produced through different methods.

As discussed previously in this dissertation, there is little consensus within the social network analysis research arena regarding the structure of real communities. There are multitudes of competing community detection algorithms and mathematical formulations of community, and different algorithms, intended to optimize for different values, can naturally produce very different communities. This is true not only for algorithms intended to find different types of mathematical structures: because optimization of many mathematical formulations is computationally intractable, even different heuristic algorithms for the same mathematical function may produce dramatically different outputs.

As an illustration of these disparities, consider Figure 5.1, which contains five communities of similar sizes from the blogging network LJ1, produced by the Metis community detection algorithm, a simple random walk with restart, Infomap, Newman-Clauset-Moore modularity optimization, and Louvain modularity optimization, as well as one annotated community identified through metadata. Immediately, structural differences are clear. The Louvain community is very dense and compact, while the random walk is much wider and looser. The annotated community seems to be between these extremes: it is not nearly as tight as the Louvain community, but is better-connected than the

Figure 5.1: Six communities produced by different methods on network LJ1



Caption: These communities have clear structural differences.

random walk community. Also observe the differences between the Newman-Clauset-Moore community and the Louvain community: both of these sets were produced by algorithms intended to optimize modularity, but their structures have obvious differences.

In a related issue, as discussed in Section 1.1, researchers have not settled on a single way of evaluating communities, and so cannot easily identify which algorithm produces the 'best' output. Furthermore, while we may sometimes have examples of "real" communities (e.g., the annotated communities used throughout this dissertation), it can be difficult to characterize these sets in the absence of negative examples.

In this section, to address these issues, we propose the Community Structure Analysis Framework (CSAF), a highly scalable machine-learning-based framework for analyzing the differences in structure for communities produced through various different methods. It simultaneously considers communities produced by many different methods, each characterized using many different features, and allows a user to gain insight into the similarities and differences between different types of communities.

Our framework begins by creating structural classes of communities by first applying different community detection algorithms to a network and then calculating important features of each community, such as diameter and edge density. We then apply a machine-learning classifier to these structural classes in order to better understand the behaviors of different structural classes.

In this chapter, we first list the networks studied, and then describe the structural classes and their associated feature space. We then describe the Community Structure Analysis Framework, and then end with a sample application demonstrating how one might use the CSAF. [1]

5.1 Datasets

In this chapter, we use networks Grad, Undergrad, HS, SC, DM, Amazon, DBLP, LJ1, and LJ2, as well as their associated metadata annotations.

[1]The work in this chapter originally appeared in [1] and [2].

5.2 Algorithm Classes

We consider 10 community detection algorithms of various types. Each of these algorithms produces some output; communities produced by the same method form a class. Because we also include the set of annotated communities in our analysis, we consider a total of eleven community classes.

Due to the vast numbers of community detection methods, it is not possible for us to consider every such method; however, we have made an effort to create a diverse set of algorithms that includes methods from important categories of algorithms. Some of our methods partition the network, while others produce overlapping communities. We have included algorithms that optimize important community metrics, such as modularity or conductance. Some of the other algorithms are not properly considered community detection algorithms at all, but produce only one "community" at a time, and require the user to specify a start-node (which we randomly select) and size (which we randomly select from the sizes of annotated communities from that network).

- Breadth-First Search (BFS): Our first method is a simple breadth-first search. In this method, for a randomly-selected starting node n and size, we begin with a community that includes only node n. In each step, we add to the set all neighbors of nodes currently in the set. We continue this procedure until the desired size is reached.
- Random Walk Without Restart (RW0): The second method that we consider is the classic random walk on a graph. Here, we again specify a starting node n and a size. A random walker begins on node n, and at each step, randomly selects a neighbor of n to transition to, until the de-

sired number of nodes have been visited.

- Random Walk With Restart (RW15): The third method is a modification of method RW0. In this method, we again perform a random walk, but at each step the random walker has a 15% chance of returning to the starting node. This method results in a set that is more tightly centered about the starting node.
- $\alpha - \beta$ (AB): An $\alpha - \beta$ community, for positive integers α,β, $\alpha \leq \beta$ is a set of nodes such that every node within the community has at least β connections within the community, and every node outside the community has at most α neighbors within the community [44]. To produce an $\alpha - \beta$ community, we begin with a BFS community produced earlier, and then perform an iterative swap procedure. In each step, we remove the node in the community that has the fewest links into the community, and add the node from outside the community that has the most links into the community. This procedure is performed until an $\alpha - \beta$ community is obtained, or until a maximum number of iterations has been reached.
- InfoMap (IM): We use the InfoMap algorithm described in Section 1.1.
- Link Communities: We use the Link Communities algorithm described in Section 1.1.
- Louvain Method for Greedy Modularity Optimization (Louvain): We use the Louvain method for greedy modularity optimization described in Section 1.1.
- Newman-Clauset-Moore Method for Modularity Optimization (Newman): In addition to the Louvain method for greedy modularity optimization, we use the heuristic algorithm for modularity optimization as introduced by Newman, Clauset, and Moore [48]. In this algorithm, each node

begins in its own community. At each step, two communities are selected for merging in such a way as to produce the greatest increase (or smallest decrease) in global modularity. In such a way, a hierarchy of communities is produced, and the resulting dendrogram is then split at the level that maximizes modularity.

- Markov Clustering Algorithm (MCL): The Markov Clustering Algorithm is a random-walk-based method that consists of two alternating steps [30]. Beginning with a normalized adjacency matrix, the first step “expands” the matrix, and the second step “inflates” the matrix. In the first step, the matrix is squared: this corresponds to flow between communities. In the second step, each element in the matrix is individually squared: this corresponds to strengthening within-group links. This process eventually converges to a stationary matrix from which communities are extracted.
- Metis: Metis [30] is a partitioning method that is a variation of the Kernighan-Lin algorithm [31]. The purpose of Metis is to partition a weighted-node network into a given number of equal node-weight sets such that the number of links between sets is minimized. We weight the nodes in such a way as to produce sets with good conductance (that is, sets that are internally well-connected with few links to the outside).

Each of these methods may produce a different number of communities of varying sizes. In our experiments, we are primarily interested in communities that are not very small and not very large; we thus only include communities of between 10 and 1000 nodes. Additionally, some of these methods may produce communities with multiple components, and we discard such sets. Naturally, other researchers may make other choices.

Because each method may produce very different numbers of communities, and our classification methods are sensitive to class balance, we over- and under-sample the different classes to ensure that they are of the same sizes. For the smaller networks (Grad, Undergrad, HS, SC, and DM), each class contains 100 communities, whereas the classes of the larger networks each contain 1000 communities. Again, other researchers may choose to have smaller or larger classes: smaller classes generally lead to lower classification accuracy and faster running time, while larger class sizes may have higher accuracy, but slower running times.

After applying each of these methods to a network and identifying annotated communities, we have eleven classes of communities.

5.3 Features

Next, for each community in each class, we calculate a feature vector.

In our work, we consider 38 community features. Other users of our framework may wish to calculate more or different features. Some of these features are calculated using only information about the internal structure of the community (i.e., the community's positioning within the larger network is not considered), while other features additionally incorporate information about nodes that are not contained in the community, but have neighbors inside the community. Several features are community-wide measures, while others measure statistics of single nodes, node pairs, or edges. For features of the latter type, we report the quartile values for that feature over all nodes, node pairs, or edges.

We calculate the following features:

- Number of nodes n in community
- Number of edges m in community
- Diameter of community
- Edge density of community: ratio of m to the maximum possible number of edges given n
- Conductance: ratio of m to the sum of the degrees of nodes (including edges going outside the community)
- Transitivity: ratio of the number of closed triangles to the number of 2-hop paths
- Triangle density: ratio of the number of closed triangles to the maximum possible number of closed triangles given n
- Shortest path quartiles: for every pair of nodes, calculate the minimum distance between the two nodes. From the list of all such values, find the lowest, 25th percentile, 50th percentile, 75th percentile, and highest value.
- Edge betweenness: for each edge, calculate the fraction of shortest paths between all pairs of nodes that use that edge. Edges with a high edge betweenness score tend to be "bottleneck" edges linking different sections of the community, as many shortest paths rely on these edges. From the list of all edge betweenness values, find the lowest, 25th percentile, 50th percentile, 75th percentile, and highest value.
- Node betweenness: similar to edge betweenness, the node betweenness of a node is the fraction of shortest paths that use that node. From the list of all node betweenness values, find the lowest, 25th percentile, 50th percentile, 75th percentile, and highest value.

- α: for each node on the fringe of the community (connected to but not contained in the community), calculate the number of in-community neighbors of that node. From the list of all α values, find the lowest, 25th percentile, 50th percentile, 75th percentile, and highest value.
- β: for each node in the community, calculate the number of in-community neighbors of that node. From the list of all β values, find the lowest, 25th percentile, 50th percentile, 75th percentile, and highest value.
- Treesum: total number of spanning trees of the community, divided by total number of spanning trees of an n-clique (calculated using Kirchoff's matrix tree theorem [43])
- Information centrality: similar to node betweenness, except instead of considering only shortest paths between nodes, consider all paths, weighted inversely with path length [59]. From the list of all information centrality values, find the lowest, 25th percentile, 50th percentile, 75th percentile, and highest value.

5.4 Class Separability

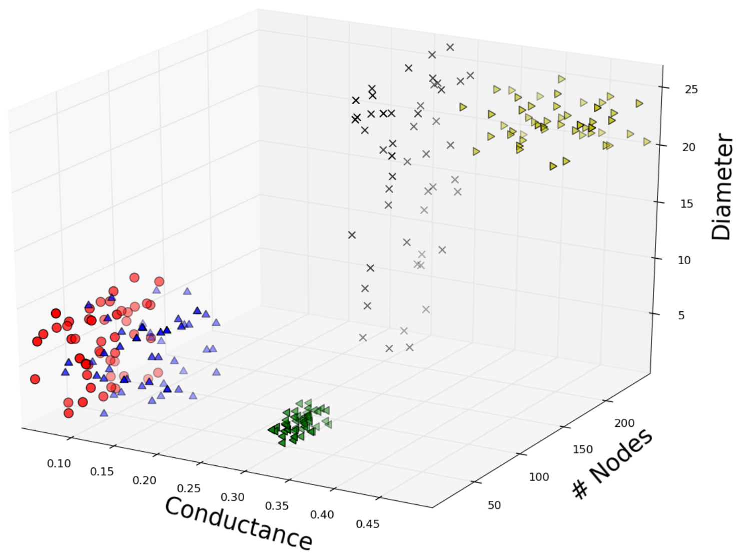
After producing the classes of communities and calculating a feature vector for each set, we have eleven classes of community feature vectors (corresponding to, for example, feature vectors for communities produced by Infomap, MCL, or annotation). With this data, we can place the classes of feature vectors in a feature space. Naturally, because we consider 38 features, we cannot visually depict this placement; Figure 5.2 contains a synthetic example of how one might do this in a much lower-dimension feature space. In this figure, there are five classes, and points of the same color represent feature vectors from the same class. In this toy example, one can immediately begin to compare different classes. For example, we see that the classes are largely separable (occupy different regions of the feature space). The classes of red circles and blue triangles, on the lower left side of the plot, have some overlap, and so are somewhat similar. The class of green triangles, at the bottom of the plot, is very compact, indicating that feature vectors in this class are very similar to one another. On the top right side of the plot, we see the classes of black crosses and yellow triangles. These two classes resemble one another more than they resemble any other class, but are still quite different. The class of black crosses is spread over a large area, suggesting that there is a lot of diversity within this class.

In the feature space, one can perform a simplistic analysis to identify which classes are fundamentally different, and which are similar.

Our initial attempts at measuring class separability use the J3 metric [61], which is based on within-class and between-class variance. To calculate the J3 criterion, one calculates two scatter matrices: the between-class scatter matrix S_m, which is simply the global covariance matrix, and the within-class scatter matrix S_w, which is the average of the covariance matrices of each class, weighted by the size of that class. The J3 score is then defined as the trace of $|S_w^{-1} S_m|$.

A high J3 score indicates that the diagonal elements of $|S_w^{-1} S_m|$ are high, so within-class covariance is low, and global covariance is high. Intuitively, a high J3 score thus suggests that each class has many similar elements, and the classes are very different from one another. We calculate the J3 separability score for the eleven classes of community feature vectors from each network in two ways:

Figure 5.2: Community feature vectors plotted in 3-dimensional feature space (synthetic example)



Caption: Separability can sometimes be observed graphically.

first, we calculate the J3 score for the classes as they actually are, and second, for the sake of calculating a baseline J3 value, we shuffle the class labels and calculate the J3 scores for the new classes. In this latter calculation, shuffling the class labels creates a dataset in which there is no separability between classes.

Table 5.1 contains the ratios of the original J3 score to the shuffled-labels J3 score. In this table, a value of 1 indicates that there is no class separability, and higher values indicate greater separability.

Even with this very simplistic measure, we can see that different algorithms produce communities with very different structures: in every network, there

Table 5.1: Ratio of actual to shuffled J3 separability scores for each network

	Grad	**Ugrad**	**HS**	**SC**	**DM**	**DBLP**	**Amaz**	**LJ1**	**LJ2**
Global Separability	**1.44**	**2.11**	**1.7**	**1.81**	**1.35**	**1.58**	**1.38**	**1.68**	**1.76**

Caption: Higher values indicate greater separability; a score of 1 indicates no separability.

is at least some class separability. However, the J3 measure is a coarse, global metric, and we have not gained insight into the behaviors of individual classes.

We are interested in a much more fine-grained understanding of class separability, class differences, and structural tendencies of each class. To this end, we present the Community Structure Analysis Framework.

5.5 Framework Outline

We present the Community Structure Analysis Framework (CSAF), a three-part supervised machine-learning-based framework for analyzing community structure. It consists of the following steps:

1. Collect and Apply Community Detection Methods: In this step, a user of the framework collects a set $A_1, A_2, ..., A_k$ of community identification methods, and applies them to the network under study. Each method A_i produces a set C_i of communities. These methods might be community detection algorithms, but one might also identify communities by using external metadata or other methods.

2. Calculate Feature Vectors: In this step, for every community in every set C_i, the user calculates a feature vector that describes the community. In our work, we consider only structural features such as number of nodes, number of edges, conductance, and so on, but depending on available data, a user might include external information as well (e.g., in a friendship network, the ratio of adults to children in the community). In this step, the user creates k classes $F_1, F_2, ..., F_k$ of feature vectors.
3. Apply a Classification Method to the Classes of Feature Vectors: In the final step, the user applies a classification algorithm to the k classes of feature vectors. Depending on how this step is performed, the learned model may be used to answer questions such as: Are the classes separable from one another (that is, are the classes sufficiently different that the classifier can distinguish between them)? In which ways are the classes different? Do any classes resemble one another (that is, do two methods tend to produce communities that have similar feature vectors)? For the results presented in this paper, we use a Support Vector Machine classifier; however, other methods may be more appropriate for different tasks.

Through application of the CSAF, one can gain a better understanding of the relationships between different classes of communities. For example, as we do in the next chapter, one might apply a cross-validation strategy in order to evaluate the classification model: can the model correctly predict the class labels of withheld community feature vectors, or are there many misclassifications? Rather than simply looking at the global accuracy (which might give results similar to the J3 metric presented earlier), one can examine the classification accuracies of each class, and identify whether any pairs of classes are often confused.

To a researcher, the CSAF can give insight into the behavior of existing community detection methods. The CSAF is also potentially of value to practitioners, and in the next section, we present a sample application of the CSAF. In the next chapter, we apply the CSAF to the datasets, algorithms, and features described earlier, and present an in-depth analysis of separability and structural tendencies of different classes.

5.6 Sample Application

This framework is relevant to both researchers and practitioners. Researchers may use it to compare and analyze the structures of communities produced by different methods, and thus better understand the behavior of different mathematical metrics and algorithms on real data. One might use the framework to, for instance, identify which ostensibly different algorithms produce very similar communities (or vice versa). It is also valuable to practitioners: to demonstrate this, we present an example application here.

Being a supervised method, the CSAF is suited for applications in which one has examples of "real" communities and wishes to select a community detection method that produces communities that are structurally similar to the "real" communities.

Consider an administrator at a residential college who is responsible for assigning first-year students into dormitory floors. The incoming class must be partitioned into non-overlapping sets (i.e., no student will live on multiple dormitory floors). The administrator has access to the students' social network (perhaps from Facebook or surveys on residency paperwork), and has reason to believe that different dormitory floor social structures lead to different outcomes in student grades: for example, perhaps a floor containing one large loosely-knit community leads to more socializing and thus lower grades, while a floor containing many small tightly-knit groups leads to fewer parties and higher grades. The administrator cannot precisely formulate which structures lead to student success, but has examples of successful communities from past years (i.e., dorm floors that had high GPAs), and through the social network data, knows the structures of those communities.

The administrator needs to partition the incoming first-year students into communities. He or she has collected a set of potential community detection algorithms that can do so, but does not know which algorithm is most likely to produce communities that have "good" structure. Through use of the Community Structure Analysis Framework, the administrator can apply each of the community detection algorithms to the student social networks from past years, and thus obtain several classes of communities. They can then convert every community in each class into a feature vector, and then train a classifier to learn the structures of communities produced by each method. Finally, the administrator can apply the classifier model to the set of "good" communities, and by observing how the model tends to classify those "good" communities, identify which community detection algorithm is most likely to produce communities that structurally resemble successful communities from past year, and thus are most likely to result in high student success.

CHAPTER 6

APPLICATION OF THE COMMUNITY STRUCTURE ANALYSIS FRAMEWORK

In this chapter, we apply the Community Structure Analysis Framework to the datasets listed in the previous chapter. As discussed earlier, a user of the CSAF may select which classification algorithm to use. In this text, we present results obtained through application of a Support Vector Machine classifier [64], as implemented through the freely-available software package LibSVM [9]. For purposes of verification, we also performed these experiments using the k-Nearest Neighbors classification method, and obtained similar results.the Community Structure Analysis Framework, and then end with a sample application demonstrating how one might use the CSAF. [1]

6.1 Background

In this chapter, we use a multi-class probabilistic Support Vector Machine (SVM) algorithm in the application of the Community Structure Analysis Framework. An SVM is a supervised learning model used for classification tasks: that is, given a set of training data points, each labeled with a class name, the SVM learns a model for predicting class labels. For example, in the CSAF, an SVM algorithm may be given many examples of feature vectors corresponding to communities produced by different methods, each labeled with the name of the method that produced the associated community. With this training data, the SVM will learn a model for differentiating between the different method-

[1]The work in this chapter originally appeared in [1] and [2].

classes (e.g., it will learn that a BFS community tends to have a specific type of feature vector, and an Infomap community tends to have a different type of feature vector). This model can then be used to classify unlabeled data: that is, we can use the model to predict which method produced the community corresponding to an unlabeled feature vector.

The simplest, and original, version of the SVM considered only two classes and did not allow for mislabeled data. The goal of such an SVM is to find a hyperplane that cleanly separates the two classes in such a way as to maximize the distance from the nearest data point to the hyperplane (i.e., to maximize the "margin"). Because this may not be possible in the original feature space, the SVM will map the data points into a high- or infinite-dimensional space, and find a separating hyperplane in this new space.

Because data is often mislabeled or noisy, a soft margin version of the above is commonly used. In this model, data points may appear on the wrong side of the separating hyperplane, but in such cases, a penalty is applied to the objective function.

A multi-class SVM is produced by combining several single-class SVMs. In the software we use, the "one vs. one" method is employed. Here, for k training classes, a single-class SVM is built for each of the $\frac{k(k-1)}{2}$ pairs of classes. To determine the class label of a data point, each of these $\frac{k(k-1)}{2}$ models is applied to that data point. The data point is then classified as the class that receives the most votes from this process.

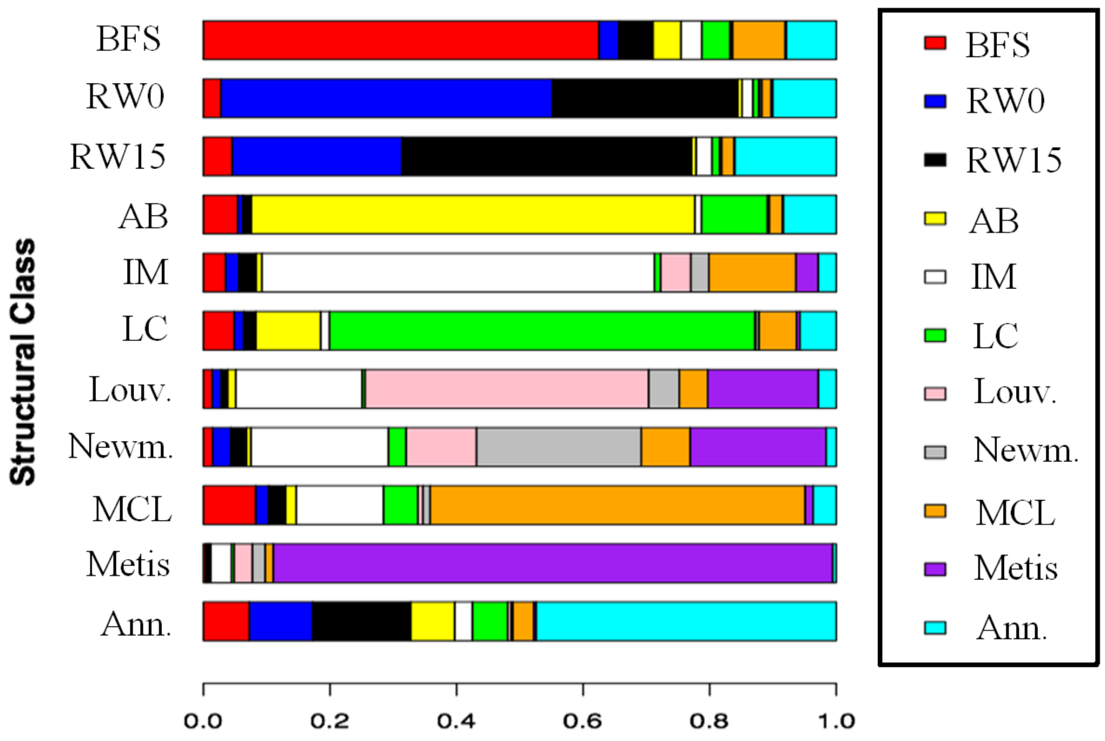
A standard SVM is non-probabilistic: that is, it simply assigns a data point to a single class. The LibSVM software that we use implements a probabilistic version of the SVM; full details are availble in [9]. In this version, rather than labeling a data point with a single class label, the model produces a probability vector $(p_1, p_2, ..., p_k)$, where $p_1 + p_2 + ... + p_k = 1$, containing the probabilities that the element belongs to each of the k classes.

6.2 Structural Consistency of Algorithm Classes

In our first and most basic set of experiments, we are interested in determining whether the eleven classes of algorithm-produced and annotated communities are internally consistent and separable from one another using the features described in the previous chapter. To address this question, for each network, we perform the following experiment:

For each network, we perform a three-fold cross-validation experiment. In each of three rounds of this experiment, we withhold one third of the data from each of the eleven classes of feature vectors for testing, and use the remaining two thirds of the data to train a Support Vector Machine. During the training phase of each round, the SVM learns a classification model, and then during the testing phase, this model is applied to the withheld data. In the training phases, the SVM learns to distinguish between the different classes of community feature vectors: for example, it learns how to differentiate between a BFS community feature vector and an annotated community feature vector. In the test phases, this model is applied to withheld data in an attempt to identify how well the model learned from the training data fits the test data. High classification accuracy during the testing phase indicates that the model learned during training is good; this indicates that the test data is similar to the training data,

Figure 6.1: Cross-validation accuracy for network DBLP



Caption: Diagonal dominance indicates high separability.

and so we can say that the classes are consistent. In contrast, if a class is structurally inconsistent, then the test data will be dissimilar to the training data, and so the model learned from the training data will produce low classification accuracy when evaluated on the test data.

Figure 6.1 contains the cross-validation accuracy results for network DBLP. In this figure, every row corresponds to a class of community feature vectors in the test sets, and the class distribution within each row describes the average of the probability vectors output by the classifier for elements from that class in the test set.

For example, the first row describes the classification probability vectors of communities in the test set that were produced by the BFS algorithm. For these communities, on average, approximately 60% of the probability mass was assigned to class BFS, 3% to class RW0, and so on. In this plot, diagonal dominance indicates high cross-validation accuracy.

We see that for most classes, cross-validation accuracy is quite high, suggesting that these classes are structurally consistent and distinct from other classes. This is especially true for certain classes such as Metis and Link Communities, indicating that these classes are especially separable from the other classes: these algorithms produce communities with structures very different from those produced by other algorithms, and so the classifier tends to assign a high fraction of the probability mass for communities in these classes to the correct class label.

In some sense, the high cross-validation accuracy that we see here is unsurprising, as we naturally expect communities produced by the same algorithm to resemble each other. The most surprising and important result here, thus, is not the performance of the model on the algorithm classes, but rather the high structural consistency of the class of annotated communities. We see nearly 50% accuracy in classifying annotated communities: while this figure is not as high as for some of the other classes, it is still quite high given that there are eleven classes. This result indicates that annotated communities share common structural features, and this structure is not adequately captured by any of the algorithms that we have considered.

In a related question, we identify which features are most valuable for distinguishing between classes, and thus gain insight into the structural tendencies of each class of communities. To address this, we apply the Correlation-based Feature Selection algorithm, a method used to identify a discriminative subset of features [27], as implemented in the machine-learning toolkit WEKA [26]. The goal of the CFS algorithm is to identify features that are highly correlated with class label, but poorly correlated with one another.

For a given subset S containing k features, CFS defines a merit function M_S.

$$M_S = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}}. \tag{6.1}$$

Here, $\overline{r_{cf}}$ represents the mean correlation between each individual feature f in S with the class label c, and $\overline{r_{ff}}$ represents the mean correlation between pairs of features in K. Sets of features that can accurately predict the class label, but are not correlated with each other, have a high M_S score. CFS begins with an initially empty set of features, and then applies a hill-climbing method to identify a set of features with a high M_S score. It may backtrack up to 5 times per iteration to search for a subset S with greater M_S.

We apply this algorithm to each of the nine network datasets, and see that in all cases, a fairly small subset S of features capture most of the structural differences between different classes of communities. Table 6.1 lists the features that tended to be contained in these subsets. Conductance and diameter were both often contained in S, and various centrality measures also tended to be important for many networks. Table 6.2 contains the cross-validation accuracy for the full and reduced sets of features for each network: we see that when using the reduced set of features, there is typically a reduction in accuracy of at most a few percentage points.

Table 6.1: Summary of the feature selection results

Network	**Grad**	**Ugrad**	**HS**	**SC**	**DM**	**DBLP**	**Amaz**	**LJ1**	**LJ2**
# Features Selected	**6**	**7**	**10**	**5**	**6**	**10**	**8**	**12**	**11**
Feature									
Conductance	1	1	1	1		1	1	1	1
Diameter		1	1	1	1	1	1	1	1
Info Centrality*	2	2	3	1	1	2	1	2	2
Node Betweenness*			2	2		2	1	5	5
Shortest Path*			1		3	2	1	1	1
β^*	1	1	1			2	1	1	1
α^*	1		1		1		2	1	
Other features	Trans-itivity	Edge Den. Tri Den							

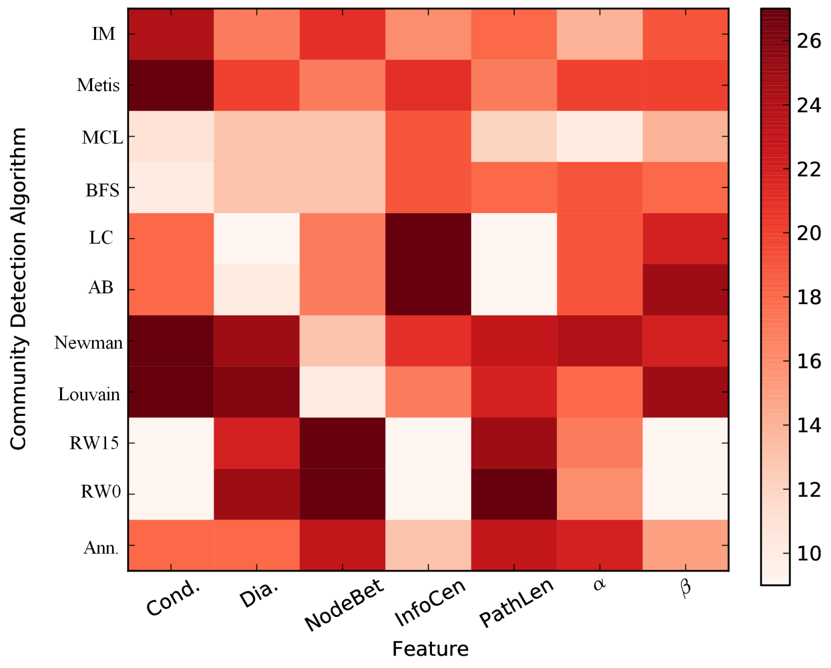
Caption: Important features for each network, as selected by CFS feature selection algorithm. For features marked with a **, multiple quartiles were calculated; the number of quartiles selected as important is contained in the table. For features not marked with a* **, only one value was calculated; if this feature was selected by CFS, the table contains the value* 1.

Table 6.2: *k*-Nearest-Neighbors classification performance using both the full and reduced sets of features

	Grad	**Ugrad**	**HS**	**SC**	**DM**	**DBLP**	**Amaz**	**LJ1**	**LJ2**
All Feats.	62.9%	86%	82.2%	80.9%	93.6%	81.3%	65.3%	89.1%	88.5%
CFS Feats.	61.5%	84.7%	85.1%	81%	90.6%	79.4%	63%	78.8%	76%

Caption: Subset of the most discriminative features is selected by CFS.

Figure 6.2: Tendency of algorithms with respect to various features



Caption: Scores are calculated by three times the number of networks on which the feature had a high score on that class plus two times the number of networks on which the feature had a medium score on that class plus the number of networks on which the feature had a low score.

Using these features, we can study the various tendencies of each class. Figure 6.2 contains seven important features, and shows whether the different classes tend to have high, medium, or low values of each of those features across the different networks. Using this figure, we can group together different algorithm classes into common groups. We see some unsurprising results, such as resemblance between the two random walk classes, as well as between the two modularity-based classes. We also see less expected results: for instance Link

Communities and $\alpha - \beta$ communities are quite similar overall, as are Infomap and Metis. As we saw earlier in this section, the annotated communities do not closely resemble any of the algorithm-based classes.

6.2.1 Structural Consistency of Algorithm Classes Across Networks

In our first experiment, we considered each network separately, and saw that within a network, communities produced by the same method tended to resemble one another. Now, we turn to the question of whether we see class consistency across different networks. To approach this topic, we perform a variation of the previous experiment, and rather than performing a cross-validation experiment on a single network at a time, perform nine experiments (one for each network), in each of which we train the SVM using examples from one network, and evaluate it on examples from all other networks.

For example, in one experiment, we train the SVM using the eleven community feature vector classes from network Grad. We then apply the learned model to examples from the other eight networks, and evaluate the model's accuracy on these examples. In this way, we learn whether, for example, a BFS community from network DM resembles a BFS community from network Grad.

Table 6.3 contains the results of this experiment. Here, each column corresponds to a different network, and the title of each column contains the name of the network on which the SVM was trained in that experiment. The value in each cell contains the classification accuracy for elements of that class from all other networks (in calculating this value, each of the eight test networks were weighted equally).

We see that some classes are much more consistent across networks than others. For example, with some exceptions, the SVM tended to correctly classify BFS, RW0, RW15, and AB elements correctly, regardless of which network it was trained on. For other classes, such as Louvain, we tend to see much lower accuracy (though again, there are some exceptions to this, such as when the SVM was trained on elements from networks Undergrad or DM).

The last row, corresponding to the class of annotated communities, is particularly interesting. As with most of the other classes, we generally see low cross-network consistency. However, as with many of the other classes, there are exceptions, such as networks HS, Amazon, LJ1, and LJ2. It is quite surprising that we see any consistency in this class; although there is some expectation that algorithms show consistency across different networks, it is shocking that externally-defined communities from fundamentally different networks (e.g., genetic vs. social) bear any resemblance to one another.

6.3 Classification of Annotated Communities into Algorithm Classes

We have seen that the structures of annotated communities are consistent and are not well-captured by any of the algorithms that we considered; nevertheless, we now attempt to determine which of these algorithms produces communities that most resemble annotated communities.

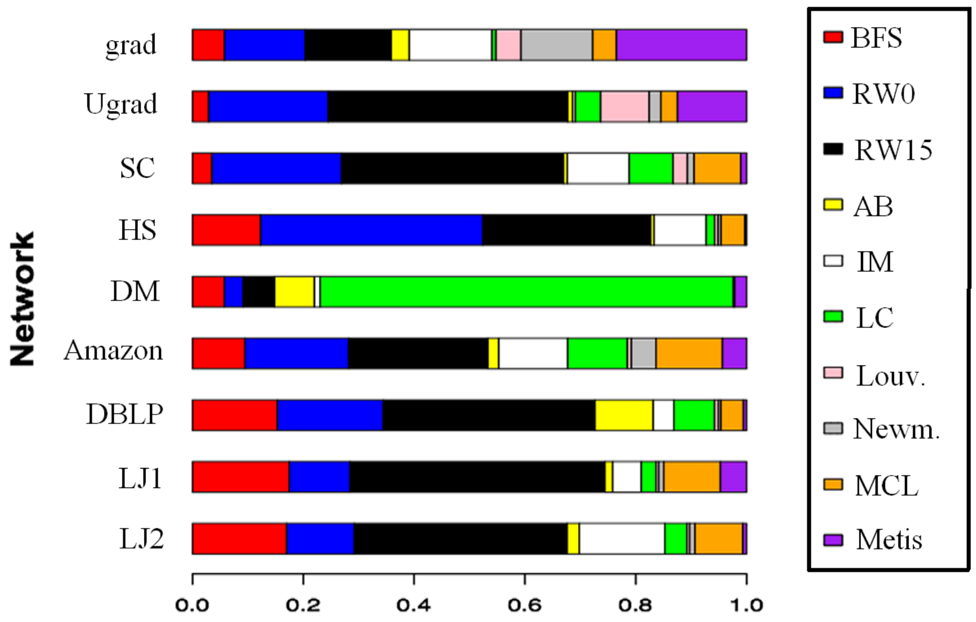
Table 6.3: The percentage of probability mass from each class that was correctly classified

Class	Grad	Ugrad	HS	SC	DM	Amaz.	DBLP	LJ1	LJ2
BFS	60.3	34.7	42.6	36.4	25.2	46.2	63.4	37.1	48.5
RW0	44.5	26.2	30.6	21.0	20.8	38.8	30.8	22.8	20.9
RW15	41.0	23.6	24.8	16.0	20.1	48.4	26.4	19.8	24.9
AB	58.4	41.7	63.6	44.5	9.9	20.0	55.1	64.3	62.8
InfoMap	4.7	5.5	22.0	42.5	23.8	6.0	25.1	34.0	33.0
LinkCom	6.6	14.6	29.4	33.8	15.6	14.8	27.7	23.2	28.4
Louvain	2.1	43.5	8.0	25.0	41.0	1.1	7.8	20.9	9.1
Newman	6.3	4.9	8.4	23.2	3.6	9.9	5.4	34.2	53.6
MCL	7.3	13.9	23.7	20.3	27.1	13.6	18.9	40.4	7.3
Metis	20.2	11.9	31.0	12.1	34.8	3.5	5.9	14.8	14.0
Ann.	5.8	19.7	23.8	10.8	20.6	31.0	17.3	36.1	34.2

Caption: The column titles indicate the networks on which the classifiers were trained. The values in row M, column N indicate the average percentage of probability mass from each class over all networks except N that was correctly classified.

To answer this question, we again apply our Community Structure Analysis Framework. In this experiment, we train the classifier using feature vectors of communities from each of the ten algorithm classes, apply the learned model to classify feature vectors of the annotated communities, and then identify which algorithm the model tends to classify the annotated communities as. As before, when classifying a particular annotated community, the classifier model outputs a probability vector containing the probabilities that that community was produced by each algorithm class.

Figure 6.3: Classification of annotated communities

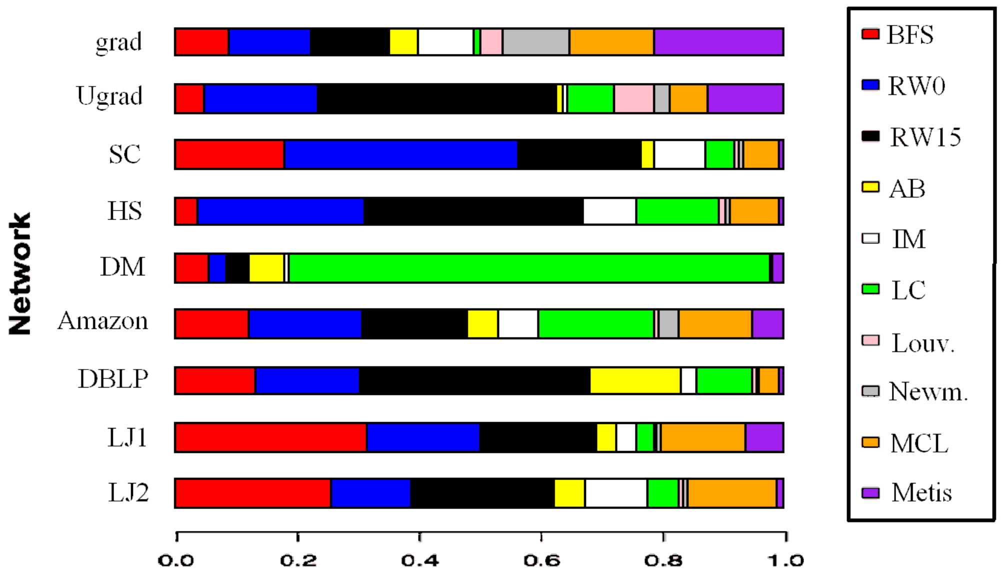


Caption: Annotated communities are generally classified as one of the two random walk methods.

Figure 6.3 shows the results of this experiment. In this plot, every row corresponds to a network, and the class distribution within each row depicts the average of the probability vectors output by the classifier for annotated communities from that network. We see that, with only two exceptions (network Grad and network DM), the annotated communities from each network tend to, on average, most strongly resemble one of the two random walk classes.

This result is particularly surprising because the random walk algorithms as implemented are extremely simple methods, especially in comparison to many of the more modern, sophisticated methods that we also included in our algorithm set (e.g., Link Communities).

Figure 6.4: Classification of annotated communities



Caption: Trimmed annotated communities are generally classified as one of the two random walk methods.

6.3.1 Classification of Annotated Community Cores

Recall that in order to identify annotated communities, we identified connected sets of nodes that had at least one feature in common. For example, we identified all graduate students in the same department, considered the subgraph induced by these nodes, and then took each connected component within this subgraph to be a separate annotated community. Because we required only simple connectivity, we allowed for the possibility of annotated communities consisting of 'tendrils' of nodes attached to a relatively dense community 'core.' For an example of such a community, refer to the annotated community in Figure 5.1: in the center of the community, we see that most nodes are fairly well-connected to one another, but there are many nodes along the fringe that have only one or two connections into the rest of the set. We might expect to see similar structure in communities generated by a random walk process, such as the random walk community displayed in Figure 5.1.

We are thus interested in examining whether the structural similarity between annotated communities and random walk communities is merely an artifact of our requiring only simple connectivity: if we had required that all nodes in annotated communities have at least two connections into the community, would we still see this resemblance?

To answer this question, we perform a trimming process on the annotated communities by iteratively removing all nodes with only one connection into the community. This process is repeated until each node in the community has at least two connections to the set. We then repeat the previous experiment, and identify which algorithm produces communities that most closely resemble the trimmed annotated communities.

Figure 6.4 contains the results of this experiment. We again see that in a majority of the networks, a plurality of the probability mass of the trimmed annotated communities is classified into one of the two random methods (though less definitively than in the previous experiment). This suggests that the similarity between random walks and annotated communities is deep and fundamental.

6.4 Network Consistency

In our final experiments, we apply the Community Structure Analysis Framework in a slightly different way, and ask whether communities from different networks can be distinguished from one another.

In the first set of experiments here, we consider each method of defining communities separately (that is, we perform a separate experiment for each of BFS, RW0, Annotated, etc.). Instead of labeling each community feature vector with the method that produced it, we label it with the network that it came from. We perform a 10-fold cross-validation experiment in which the SVM trains a model using examples from each of the nine networks (labeled with the network name), and then attempts to predict which network the withheld community feature vectors came from.

For example, in our first experiment, we consider only BFS communities. The SVM is trained on BFS community feature vectors from networks Grad, Undergrad, HS, and so on, each labeled with the network name. We then calculate the accuracy of the resulting model by evaluating how well it predicts which network each of the withheld community feature vectors came from.

In this way, we gain insight into whether different networks have fundamentally different structures from one another: e.g., we saw earlier that BFS sets had structural similarities across networks, but are there also important structural differences between BFS communities from different networks?

Table 6.4 contains the results of these experiments. Here, each row corresponds to a different experiment, and the row label contains the name of the method that produced the communities that we considered. Each cell contains the cross-validation classification accuracy for community feature vectors from the specified network. We see very high accuracy scores in general, with the exception of communities produced by the Newman-Clauset-Moore modularity algorithm (this may be because for many networks, this algorithm produced a very small number of communities, resulting in a very small training set and thus a less accurate SVM model). Interestingly, while we saw from earlier experiments that the class of annotated communities was somewhat distinguishable from the algorithm classes, even across different networks, we see here that annotated communities from different networks also have fundamental dissimilarities. (Note that these results are not contradictory: the earlier experiment demonstrated that annotated communities across networks have important similarities, and we see here that there are also important differences).

Next, we perform a similar experiment, but instead of evaluating each type of community separately, we conduct a single cross-validation experiment in which we consider all eleven methods of identifying communities separately. Table 6.5 contains the results of this experiment. Here, the row labels contain the names of the networks that community feature vectors actually belong to, and the column labels contain the names of the networks that they were classified as. For instance, of community feature vectors that are from network Grad, an average of 0.5% of their probability mass was classified as network Undergrad. In this table, diagonal dominance indicates high accuracy.

We see that, as in the previous experiment, communities from different networks tend to be separable from one another. We can interpret these results as a network similarity matrix; if two networks are frequently confused with one

Table 6.4: Network classification accuracy, training and test sets contain elements from one algorithm class

Class	Grad	Ugrad	HS	SC	DM	Amazon	DBLP	LJ1	LJ2
BFS	68.9%	67.4%	36.6%	25.0%	61.3%	78.6%	71.0%	47.3%	43.3%
RW0	67.5%	74.2%	45.1%	49.5%	68.2%	82.8%	74.1%	55.4%	51.0%
RW15	69.3%	76.2%	47.6%	50.1%	77.2%	83.7%	76.0%	55.5%	49.2%
AB	69.8%	55.1%	30.5%	31.2%	72.0%	82.1%	70.9%	43.4%	40.4%
InfoMap	33.2%	87.8%	47.6%	51.9%	54.7%	82.1%	78.3%	62.3%	57.1%
LinkCom	38.8%	94.2%	72.9%	66.7%	74.6%	81.1%	78.1%	54.1%	53.5%
Louvain	50.3%	71.9%	72.4%	46.4%	2.6%	90.1%	88.3%	39.9%	48.5%
Newman	0.2%	0.1%	0.3%	0.1%	0.1%	94.8%	21.5%	15.2%	60.8%
MCL	20.0%	48.1%	49.9%	22.3%	22.4%	80.6%	72.5%	53.0%	55.8%
Metis	92.3%	98.1%	88.5%	83.9%	81.5%	98.2%	98.0%	92.0%	92.1%
Ann.	60.4%	52.8%	22.4%	27.3%	38.3%	91.8%	85.7%	44.5%	45.8%

Caption: The percentage of probability mass from each network that was correctly classified as belonging to that network. The row titles indicate the class that the classifier was trained on.

another by the SVM, then this indicates that their communities are structurally similar. We see that, for instance, networks LJ1 and LJ2 are often confused with one another: this is to be expected, because they are simply different portions of the same larger network. We see some much more surprising results. For example, although networks Grad and Undergrad are both Facebook networks for students from the same university, there is very little confusion between the two: communities from these two networks are very different from one another.

In our final experiments of this nature, we give the classifier a much more difficult task. In the last two sets of experiments, the training data contained at

Table 6.5: Network classification accuracy, training and test sets contain elements from all algorithm classes

	Grad	**Ugrad**	**HS**	**SC**	**DM**	**Amazon**	**DBLP**	**LJ1**	**LJ2**
Grad	62.5%	0.5%	6.0%	1.8%	0.6%	6.9%	8.3%	4.5%	8.8%
Ugrad	3.7%	74.0%	2.0%	9.0%	1.4%	0.4%	0.9%	5.0%	7.0%
HS	2.3%	1.6%	44.2%	7.1%	2.0%	13.0%	5.9%	11.4%	12.6%
SC	1.6%	4.9%	6.3%	42.8%	4.3%	2.1%	4.7%	17.7%	15.5%
DM	0.5%	2.2%	4.1%	6.5%	60.7%	1.2%	3.1%	11.7%	9.9%
Amazon	1.8%	0.3%	4.6%	0.8%	0.4%	71.6%	11.3%	3.4%	5.3%
DBLP	2.4%	0.5%	3.0%	2.1%	0.9%	12.5%	64.1%	6.1%	8.1%
LJ1	2.0%	1.9%	6.4%	7.7%	4.3%	4.8%	6.8%	35.8%	30.3%
LJ2	2.8%	2.2%	6.1%	7.4%	3.9%	5.1%	8.1%	27.3%	37.3%

Caption: The percentage of probability mass from elements in each network that was classified as each network. Row $N1$, Column $N2$ contains the fraction of probability mass of elements from network $N1$ in the test set that was classified as network $N2$.

least some communities produced by the same method as the ones being classified: either the training and test sets contained only communities produced by the same method (e.g., we considered only BFS communities), or the training and test sets both contained many communities of all types (e.g., BFS, Infomap, Annotated). Here, we perform eleven separate experiments. In each, we train the classifier on communities produced by a single method, again labeled with the name of the network from which they came, and test the classifier on communities produced by all other methods.

For example, we perform one experiment in which the classifier is trained on BFS communities from all networks, labeled with the network names. The classifier then attempts to predict the network labels of communities produced

Table 6.6: Network classification accuracy, training and test sets contain different types of communities

Training Class	Grad	Ugrad	HS	SC	DM	Amazon	DBLP	LJ1	LJ2
BFS	43.5%	11.5%	11.0%	9.7%	79.2%	73.9%	45.4%	31.2%	5.2%
RW0	40.0%	13.1%	11.0%	11.5%	68.5%	76.0%	44.1%	22.7%	7.3%
RW15	35.9%	14.7%	10.9%	15.2%	48.8%	76.6%	47.7%	19.5%	9.2%
AB	30.5%	12.0%	8.7%	12.6%	44.3%	73.4%	46.8%	18.2%	11.2%
InfoMap	25.4%	12.7%	9.0%	19.2%	49.6%	66.0%	44.8%	17.5%	14.2%
LinkCom	21.8%	11.7%	14.4%	21.8%	49.1%	60.4%	44.2%	15.1%	12.2%
Louvain	19.2%	10.6%	13.4%	19.8%	43.2%	52.8%	43.9%	15.8%	11.3%
Newman	17.3%	10.3%	12.0%	17.1%	37.1%	50.3%	40.3%	18.1%	16.8%
MCL	16.4%	11.1%	14.2%	18.2%	32.7%	49.0%	39.9%	19.8%	16.4%
Metis	16.8%	11.4%	14.6%	16.7%	39.2%	45.9%	37.0%	20.3%	16.8%
Ann.	28.6%	20.5%	10.9%	12.8%	60.4%	60.3%	41.7%	22.3%	16.8%

Caption: The percentage of probability mass from each network that was correctly classified as belonging to that network, where the training set contained elements from the specified class.

by other methods: can it identify which network a community came from, even when that community was produced through a method fundamentally different from the ones that produced the communities on which the classifier was trained?

Table 6.6 contains the results of these experiments. Here, each row corresponds to a separate experiment. The row labels contain the name of the method used to produce the communities in the training set, and the value in each cell is the accuracy of the resulting classification model when predicting network labels of communities produced through other methods. For example, when an SVM is trained using communities produced through a BFS, it has an average 43.5% accuracy at predicting which communities, produced through other methods, came from network Grad.

We see that some networks, such as Grad, DM, Amazon, and DBLP, have an extremely distinctive signature. An SVM classifier has very high accuracy rates at identifying which communities came from these networks, even when the training communities are fundamentally different from the test communities.

6.5 Conclusion and Future Work

The social network analysis literature contains many different community detection algorithms; in nearly every case, the behavior of an specific algorithm is not well understood. To address this issue, in Chapters 5 and 6 of this dissertation, we introduced and applied the Community Structure Analysis Framework. The CSAF is a machine-learning-based system that allows both users and practitioners to understand and compare the behaviors of different community detection algorithms and select the best algorithm for a specific application and network.

When using the CSAF, one first collects a set of *k* community detection algorithms and applies them to a network to produce *k* classes of communities. For each community, one then calculates a feature vector containing important characteristics of that community, such as size, conductance, and diameter, and thus obtains *k* classes of community feature vectors. By applying an SVM (or other type of classification method), one then learns a model to distinguish between the different classes. This model can be used to understand important characteristics of each class (e.g., one community detection method may produce communities that are large and have low conductance), analyze the internal consistency of each class, or identify pairs of community detection algorithms that produce very similar communities.

We used the CSAF to study a collection of 11 classes of community identification methods (including 10 classes of algorithms and one class of annotated communities), 38 community features, and 9 networks. We obtained several interesting results, including the conclusion that the class of “real” annotated communities had their own consistent structure that was distinct from the structures of communities produced by algorithms. We saw that most algorithms tended to produce consistent structure, even across different networks, and also saw that communities from each network possessed a distinctive signature.

In future work, we are interested in using the CSAF to better understand the structure of annotated communities. We saw in this chapter that annotated communities tend to have a consistent structure, and we wish to better understand the details of this structure. In addition, we are studying the possibility of using the CSAF to build a supervised community detection algorithm: in such an algorithm, a user would provide examples of “good” communities, the CSAF would learn a model for these communities, and then search a network to find other sets with similar structures.

CHAPTER 7

CONCLUSION

The tools and techniques developed by researchers in the field of computational social network analysis have fundamentally altered the way we study and understand real networks. An area that is of particular interest is the study of communities. Although an understanding of community structure is critical to understanding networks as a whole, researchers within the field have not arrived at a consensus regarding how to properly characterize community structure.

In this dissertation, we have presented several important contributions to community research.

We began with an introduction to the area, an overview of related work, and a description of the datasets that we used throughout our work.

In Chapter 3, in order to motivate the topic, we presented a simple application of community detection to the link prediction problem. In this problem, one is given incomplete network data, and must predict which missing links in the incomplete data are most likely to exist in the actual network. We showed that for simple link prediction metrics, incorporation of community membership information could substantially increase the performance of those metrics. In this chapter, we considered a variety of similarity based link prediction methods, such as the Common Neighbors metric, which calculates the number of shared neighbors that two nodes have, and predicts that nodes with many neighbors in common are most likely to be connected. We modified such metrics in ways that used node community membership information; for example, if two nodes and a common neighbor were all in the same community, then this shared neighbor was weighted more heavily than shared neighbors in different communities from the two nodes being considered. Such modifications typically resulted in large increases in link prediction accuracy over a variety of networks.

While the study of communities is important to a range of topics both within and outside of computer science, it has been complicated by difficulties in characterizing the structure of communities. Scientists in this area have developed a wide range of mathematical measures of community, many based on the general principle that a good community is well-connected internally and mostly separate from the rest of the network. Some of these metrics, such as modularity and conductance, are based in the principle that communities ought to be “round”, or internally well-connected throughout, like a clique or a $G(n, p)$ graph. In contrast, other notions of community allow for “long” communities that are formed of small, well-connected groups that are connected to one another.

In Chapter 4, to study the validity of these two competing notions, we examined the structure of annotated communities identified through network metadata. Examples of such communities included the set of all students in the same department, or the set of all books by the same author. For each of these annotated communities, we compared the diameter of that community to the diameter of a random $G(n, p)$ graph with the same number of nodes and edges. We saw that the annotated communities tended to have significantly higher dimaeters than the random graphs of the same size, and concluded that the “round” model was not appropriate for these communities. We then decomposed each of these communities into smaller sub-communities, and found that these sub-communities *did* tend to be “round”; moreover, the graph representing the connections between these sub-communities was *also* "round." We thus concluded that while real communities did not fully fit the "round" model themselves, they could be represented as "round" sub-communities connected in a "round" manner.

With this intuition in mind, we presented the Node Perception algorithm template for community detection. This three-part method begins by considering each node separately, and finding "sub-communities" centered about that node. This is done by, for each node n, considering the subgraph induced by the neighbors of n, and applying a community detection method to that subgraph. This produces some number of sets, to each of which we add node n to obtain the "sub-communities." In the next step, we make a new network in which every node represents a sub-community, and two nodes are linked if their corresponding sub-communities have some amount of overlap. Finally, we apply a community detection algorithm to this new network, and expand of the resulting sets into communities from the original network. We show that this algorithm outperforms several other state-of-the-art algorithms at the task of recovering annotated communities over a wide variety of networks.

Finally, in Chapters 5 and 6, in order to further examine the structure of communities, we presented and applied the Community Structure Analysis Framework. In this machine-learning-based framework, one creates several classes of communities, each containing communities produced by a different method (e.g., an algorithm or metadata). For example, we created a class of communities produced through a breadth-first-search, another class of communities produced through the Infomap community detection method, a third class of communities identified through network metadata, and so on. For each community, one then calculates a feature vector that includes characteristics such as community size, diameter, or conductance, and in doing so, creates several classes of feature vectors. Using these classes of feature vectors, one can train a classifier model to better understand the similarities and differences between different classes. One might ask, for example, whether the classifier tends to confuse two classes: this would suggest that those two classes tend to have communities with similar structure.

We then applied the Community Structure Analysis Framework to several networks, each with metadata allowing us to identify annotated communities. We obtained several interesting results. We saw that different algorithms tend to produce very different community structures: that is, when evaluated via cross-validation, the classifier model tended to accurately classify communities, suggesting that each class of communities had a fundamentally different structural profile than the other classes. Interestingly, the class of annotated communities was also structurally consistent, and was not generally confused with any other class, suggesting that none of the algorithms that we considered correctly captured the structure of real communities. However, of the algorithms that we considered, a very simple random-walk-based method tended to most closely capture the structure of real communities.

We believe that the Community Structure Analysis Framework will be of great use to practitioners, by allowing them to select an appropriate community detection algorithm for their application, as well as to researchers, by permitting a better understanding of the behavior of different algorithms.

Research in the field of social network analysis has contributed important tools for analyzing and understanding the real networks that exist all around us.

An especially active area of study within the field deals with detecting and characterizing communities within networks. Here, we presented an application to demonstrate the need for community research, analyzed the structure of real communities and then described a novel community detection algorithm based on these results, and finished with the presentation of a machine-learning-based framework for understanding community structure. Through these contributions, this dissertation has illustrated and tied together several important facets of community research.

BIBLIOGRAPHY

[1] Bruno Abrahao, Sucheta Soundarajan, John Hopcroft, and Robert Kleinberg. On the separability of structural classes of communities. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pages 624–632. ACM, 2012.

[2] Bruno Abrahao, Sucheta Soundarajan, Robert D. Kleinberg, and John E. Hopcroft. A separability framework for analyzing community structure. *ACM Transactions on Knowledge Discovery from Data: Special Issue on Computational Aspects of Social and Information Networks*, 2013 (forthcoming: accepted with minor revisions).

[3] Balász Adamcsek, Gergely Palla, Illés Farkas, Imre Derényi, and Tamás Vicsek. Cfinder: locating cliques and overlapping modules in biological networks. *Bioinformatics*, January 2006.

[4] Yong-Yeol Ahn, James P. Bagrow, and Sune Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466(7307):761–764, 2010.

[5] Luis A. Amaral. A truer measure of our ignorance. *Proceedings of the National Academy of Sciences*, 105(19):6795–6796, May 2008.

[6] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group formation in large social networks: Membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2006.

[7] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics*, March 2008.

[8] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

[9] Chih C. Chang and Chih J. Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3), May 2011.

[10] Aaron Clauset, Cristopher Moore, and Mark E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, 2008.

[11] Aaron Clauset, Cosma Rohilla Shalizi, and Mark E. J. Newman. Power-law distributions in empirical data, 2007. cite arxiv:0706.1062 Comment: 43 pages, 11 figures, 7 tables, 4 appendices; code available at http://www.santafe.edu/ aaronc/powerlaws/.

[12] Michele Coscia, Giulio Rossetti, Fosca Gianotti, and Dino Pedreschi. Demon: a local-first discovery method for overlapping communities. In *Proceedings of the 18th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2012.

[13] Robert L. Cross, Andrew Parker, and Robert Cross. *The Hidden Power of Social Networks: Understanding How Work Really Gets Done in Organizations*. Harvard Business School Press, 1st edition, June 2004.

[14] Émile Durkheim. *The Rules of Sociological Method. Translated by S. A. Solovay and J.H. Mueller*. The Free Press, New York, 1950[1895].

[15] Paul Erdös and Alfréd Rényi. On random graphs i. *Publ. Math. Debrecen*, 6:290, 1959.

[16] Leonard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1741.

[17] Santo Fortunato. Community detection in graphs. *Phys. Reports 486*, pages 75–174, June 2010.

[18] Santo Fortunato and Marc Barthelemy. Resolution limit in community detection. In *Proceedings of the National Academy of Sciences of the United States*, 2006.

[19] Adrien Friggeri, Guillaume Chelieu, and Eric Fleury. Egomunities, exploring socially cohesive person-based communities. *CoRR*, 2011.

[20] Michelle Girvan and Mark E. J. Newman. Community structure in social and biological networks. *Proc. of the National Academy of Sciences*, 99(12):7821 –7826, June 2002.

[21] David F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pages 597–605, New York, NY, USA, 2012. ACM.

[22] Mark S. Granovetter. The Strength of Weak Ties. *The American Journal of Sociology*, 78(6):1360–1380, 1973.

[23] Jonathan L. Gross and Jay Yellen, editors. *Handbook of Graph Theory (Discrete Mathematics and Its Applications)*. CRC Press, 1 edition, December 2003.

[24] Roger Guimerà, Leon Danon, Albert Guilera-Díaz, Francesc. Giralt, and Alex Arenas. Self-similar community structure in a network of human interactions. *Physical Review E*, 68(6):065103+, December 2003.

[25] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[26] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[27] Mark A. Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, Department of Computer Science, University of Waikato, 1999.

[28] Paul Jaccard. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

[29] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, May 2004.

[30] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20:359–392, December 1998.

[31] Brian W. Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(1):291–307, 1970.

[32] G. R. Kiss, C. Armstrong, R. Milroy, and J. Piper. An associative thesaurus of English and its computer analysis. In A. J. Aitkin, R. W. Bailey, and N. Hamilton-Smith, editors, *The computer and literary studies*. University Press, Edinburgh, UK, 1973.

[33] Andrea Lancichinetti and Santo Fortunato. Community detection algorithms: A comparative analysis. *Physical Review E*, 80:056117, Nov 2009.

[34] Andrea Lancichinetti, Filippo Radicchi, Jos J. Ramasco, and Santo Fortunato. Finding statistically significant communities in networks. *PLoS ONE*, 6(4):e18961, 04 2011.

[35] Elizabeth A. Leicht, Petter Holme, and Mark E. J. Newman. Vertex similarity in networks. *Physical Review E*, 73(2):026120+, February 2006.

[36] Jure Leskovec, Lada Adamic, and Bernardo Huberman. The dynamics of viral marketing. In *Proceedings of the 7th ACM Conference on Electronic Commerce*, 2006.

[37] Jure Leskovec, Deepayan Chakrabarti, Jon M. Kleinberg, and Christos Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In Alpio Jorge, Lus Torgo, Pavel Brazdil, Rui Camacho, and Joo Gama, editors, *PKDD*, volume 3721 of *Lecture Notes in Computer Science*, pages 133–145. Springer, 2005.

[38] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 641–650, New York, NY, USA, 2010. ACM.

[39] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1361–1370, New York, NY, USA, 2010. ACM.

[40] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007.

[41] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 695–704, New York, NY, USA, 2008. ACM.

[42] Linyuan Lu and Tao Zhou. Link prediction in complex networks: A survey. *Physica A*, 390(6):11501170, 2011.

[43] Russell Lyons and Yuval Peres. *Probability on Trees and Networks*. Cambridge University Press, 2012.

[44] Nina Mishra, Robert Schreiber, Isabelle Stanton, and Robert Tarjan. Finding strongly knit clusters in social networks. *Internet Mathematics*, 5(1):155–174, January 2008.

[45] Alan Mislove, Bimal Viswanath, Krishna Gummadi, and Peter Druschel. You are who you know: Inferring user profiles in online social networks. In *Proc. 3rd ACM Intl. Conf. on Web Search and Data Mining*, 2010.

[46] Mark E. J. Newman. Clustering and preferential attachment in growing networks. Working Papers 01-03-021, Santa Fe Institute, March 2001.

[47] Mark E. J. Newman. Assortative mixing in networks. *Phys. Rev. Lett.*, 89:208701, 2002.

[48] Mark E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577 –8582, June 2006.

[49] Gergely Palla, Imre Derenyi, Illes Farkas, and Tamas Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, June 2005.

[50] Daniel Park, Rohit Singh, Michael Baym, Chung-Shou Liao, and Bonnie Berger. IsoBase: a database of functionally related proteins across PPI networks. *Nucleic Acids Research*, 39(suppl 1):D295–D300, 2011.

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

[52] Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

[53] Jar Saramäki, Mikko Kivelä, Jukka-Pekka Onnela, Kimmo Kaski, and János Kertesz. Generalizations of the clustering coefficient to weighted complex networks. *Physical Review E*, 75(2):027105, 2007.

[54] Georg Simmel. *Sociological Theory*. McGraw-Hill, 7th edition, 2008.

[55] Rohit Singh, Jinbo Xu, and Bonnie Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proceedings of the National Academy of Sciences*, 105(35):12763–12768, September 2008.

[56] Thorvald Sørensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. *Biol. Skr.*, 5:1–34, 1948.

[57] Sucheta Soundarajan and John E. Hopcroft. Using community information to improve the precision of link prediction methods. In Alain Mille, Fabien L. Gandon, Jacques Misselis, Michael Rabinovich, and Steffen Staab, editors, *Proceedings of 21st ACM International Conference on World Wide Web (Companion Volume)*, pages 607–608. ACM, 2012.

[58] Sucheta Soundarajan and John E. Hopcroft. Use of local group information to identify communities in networks. *ACM Transactions on Knowledge Discovery from Data*, 2013 (forthcoming: accepted with minor revisions).

[59] Karen Stephenson and Marvin Zelen. Rethinking centrality: Methods and examples. *Social Networks*, 11(1):1–37, March 1989.

[60] Michael Stumpf, Thomas Thorne, Eric Silva, Ronald Stewart, Hyeong An, Michael Lappe, and Carsten Wiuf. Estimating the size of the human interactome. 105:6959–64+, 2008.

[61] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Academic Press, 4th edition, November 2008.

[62] Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. *Sociometry*, 32:425–443, 1969.

[63] Luc Vandendorpe. UCL- success stories. http://www.uclouvain.be/en-416677.html. Accessed: 03/11/2013.

[64] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1st edition, September 1998.

[65] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[66] Harrison C. White, Scott A. Boorman, and Ronald L. Breiger. Social structure from multiple networks. *American Journal of Sociology*, 81:730–780, 1976.

[67] Tao Zhou, Linyuan Lu, and Yi-Cheng Zhang. Predicting missing links via local information. *Eur. Phys. J. B*, 71:623–630, 2009.