Use of Local Group Information to Identify Communities in Networks

SUCHETA SOUNDARAJAN and JOHN E. HOPCROFT, Cornell University

The recent interest in networks has inspired a broad range of work on algorithms and techniques to characterize, identify, and extract communities from networks. Such efforts are complicated by a lack of consensus on what a "community" truly is, and these disagreements have led to a wide variety of mathematical formulations for describing communities. Often, these mathematical formulations, such as modularity and conductance, have been founded in the general principle that communities, like a $G(n, p)$ graph, are "round," with connections throughout the entire community, and so algorithms were developed to optimize such mathematical measures. More recently, a variety of algorithms have been developed that, rather than expecting connectivity through the entire community, seek out very small groups of well-connected nodes and then connect these groups into larger communities. In this article, we examine seven real networks, each containing external annotation that allows us to identify "annotated communities." A study of these annotated communities gives insight into why the second category of community detection algorithms may be more successful than the first category. We then present a flexible algorithm template that is based on the idea of joining together small sets of nodes. In this template, we first identify very small, tightly connected "subcommunities" of nodes, each corresponding to a single node's "perception" of the network around it. We then create a new network in which each node represents such a subcommunity, and then identify communities in this new network. Because each node can appear in multiple subcommunities, this method allows us to detect overlapping communities. When evaluated on real data, we show that our template outperforms many other state-of-the-art algorithms.

Categories and Subject Descriptors: I.5.3 [**Pattern Recognition**]: Clustering

General Terms: Algorithms, Design

Additional Key Words and Phrases: Social networks, communities

ACM Reference Format:
Sucheta Soundarajan and John E. Hopcroft. 2015. Use of local group information to identify communities in networks. ACM Trans. Knowl. Discov. Data 9, 3, Article 21 (April 2015), 27 pages.
DOI: http://dx.doi.org/10.1145/2700404

1. INTRODUCTION

As networks become increasingly prevalent in our world, researchers seek new methods to analyze them. One important area of network research deals with identifying communities in networks [Fortunato 2010; Girvan and Newman 2002]. This area has roots in the classic problem of graph clustering, but has evolved to match our intuitions about what "real" communities should look like.

Traditionally, researchers have approached this 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

This work is supported by AFOSR grant FA9550-09-1-0675.
Authors' addresses: S. Soundarajan, Department of Computer Science, Rutgers University, Piscataway, NJ 08854; email: sucheta@cs.cornell.edu; J. E. Hopcroft, Department of Computer Science, Cornell University, Ithaca, NY 14853; email: jeh@cs.cornell.edu.

DOI: http://dx.doi.org/10.1145/2700404

advantages, it is not always clear whether a particular mathematical definition of "community" is correct.

Many such mathematical definitions are based on 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 edge density of the rest 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 these 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 [Palla et al. 2005] 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 characterize annotated 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 "subcommunities" 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 subcommunities (e.g., "my coworkers," "my classmates," etc.). These subcommunities 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 subcommunity. Each node from the original network can be represented by multiple subcommunities, so a node can appear in many different communities. Because a practitioner may choose how to identify subcommunities, how to create a network of subcommunities, 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 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 DEMON algorithm [Coscia et al. 2012], 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 far more 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 article is organized as follows: first, we discuss work related to ours, including descriptions of other algorithms that we use for comparison. Next, we discuss our method in detail. After that, we discuss each of the datasets used for evaluation. 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.

2. 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 [Newman 2006], which measures the number of edges within a community as compared to the expected number if the edges had been distributed randomly, and conductance [Kannan et al. 2004], which measures the ratio of the number of edges outgoing from a set to the total number of edges incident to vertices in 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. Some researchers, such as Friggeri et al. in Friggeri et al. [2011], identify local sets that are similar to our "subcommunities," and then expand these "egomunities" into larger communities so that a particular mathematical feature of the communities is maximized. 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 [Leskovec et al. 2005].

Unlike many of these algorithms (such as the egomunities method), 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 "subcommunities."

Most similar to our work is the DEMON algorithm [Coscia et al. 2012], which can be viewed as a specific instance of our template.

In this article, we compare our method to several other algorithms, which we discuss in further detail in the following. We first discuss the Louvain method for greedy modularity optimization and Infomap, two methods for partitioning the network. The Link Communities, DEMON, and Clique Percolation algorithms are the most similar to our algorithm in principle, and so we also consider them, as well as OSLOM, another algorithm for identifying overlapping communities.

2.1. Methods for Partitioning a Network

2.1.1. Modularity (Mod). A classic method for calculating the quality of a partition is modularity, which is based on the principle that a good community is strongly connected internally and is isolated from the rest of the network. 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 (in our networks, this value is either 0 or 1), 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 [Newman 2006]. Then the modularity Q of a partition is

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

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 will thus contribute heavily to the total modularity of the partition.

In this article, we use the recursive Louvain method for greedy modularity optimization [Blondel et al. 2008].

2.1.2. Infomap (IM). We also consider the Infomap partitioning algorithm of Rosvall and Bergstrom, which views a network as an analog to a geographical map [Rosvall and Bergstrom 2008]. Rosvall and Bergstrom consider the problem of describing random walks along the network using 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."). The goal of the Infomap partitioning algorithm is to identify clusters so as to minimize the expected length of a random walk's description that uses nodes' names. 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 [Lancichinetti and Fortunato 2009].

2.2. Methods for Finding Overlapping Communities

2.2.1. Clique Percolation (CP). One common method for locating overlapping communities is the Clique Percolation method [Palla et al. 2005]. For a specified k, this method first locates all k-cliques in the network and then "rolls" together adjacent cliques. Two cliques are considered adjacent if they share $k-1$ nodes. A community is then 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. To identify Clique Percolation communities, we use the software presented in Adamcsek et al. [2006].

2.2.2. Link Communities (LC). Another successful method for finding overlapping communities is the Link Communities method [Ahn et al. 2010]. 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.

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

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

2.2.3. OSLOM. ("Order Statistics Local Optimization Method" (OSLOM) is a clustering method intended to detect overlapping, hierarchical community structure and distinguish meaningful community structure from artificial communities that occur even in random networks [Lancichinetti et al. 2011].

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 insignificant 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. For full details, see Lancichinetti et al. [2011].

2.2.4. DEMON. Democratic Estimate of the Modular Organization of a Network (DEMON) is a method for identifying overlapping communities [Coscia et al. 2012]. It is a specific instance of the general template that we present in this article, and was

created independently of, and simultaneously with, this work. We discuss it in greater detail in Section 6.

3. DATASETS

Throughout this article, we analyze seven networks of various types, each with individual nodes tagged with metadata. With this metadata, we identify "annotated communities" by grouping together nodes with the same tags. We consider only those annotated communities that contain connected components of size at least 10. If some annotated community contains multiple components that are of size 10 or larger, we consider each to be a separate community. In this section, we describe each dataset and some of its properties.

"Amazon" is a product copurchasing network from Amazon.com [Leskovec et al. 2006]. 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 *Homo Sapiens* (human), *Drosophila* (a fruit fly species), and *Saccharomyces cerevisiae*(a type of yeast) [Park et al. 2011; Singh et al. 2008]. 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 "Ugrad" are, respectively, sections of the Facebook network that correspond to graduate and undergraduate students at Rice University [Mislove et al. 2010]. Grad contains 503 nodes and 3256 edges, and Ugrad contains 1220 nodes and 43,208 edges. For each graduate student, we are given their 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 [Cross et al. 2004]. 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.

See Yang and Leskovec [2012] for a discussion of similar annotated communities. This work demonstrates, among other results, that when such communities are perturbed through various methods (e.g., swapping nodes in and out of the community), the perturbed community tends to be a weaker community, as measured through different community evaluation methods (such as conductance). Also see Abrahao et al. [2012] for a machine-learning-based analysis of annotated communities, showing that the class of annotated communities has some amount of structural cohesion.

4. COMMUNITY STRUCTURE

As we saw in the Related Work section, 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 [Erdős and Rényi 1959].[1] 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." For instance, consider a community found by Clique Percolation that was identified by connecting adjacent k-cliques $C_1, C_2, \ldots, C_r$, 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 both models claim that communities consist of small groups. Consider, for example, a community found by Clique Percolation, which is clearly made of small groups, but may have no hierarchical structure.

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.

4.1. Community Roundness

To study community roundness, it is tempting to use existing, well-understood community measures that are based on the principle that communities ought to be "round," such as conductance or modularity. However, these metrics often incorporate other beliefs about community structure; for instance, both conductance and modularity score more highly in those communities that are well separated from the rest of the network. Moreover, conductance and modularity statistics can be difficult to interpret without a baseline for comparison. To address these issues, in this section we present the "roundness" statistic. This statistic is intended to measure only the roundness of a community without complication from other factors, and explicitly incorporates a baseline so that values can be understood in isolation.

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 expected 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. Observe that a community with a clear hierarchy may have a high "roundness" score (that is, it may be classified as "long"), because it contains regions with higher density than the community as a whole. Although such a community is clearly structurally different from an archetypal "long" community such as a path graph, we nevertheless consider it to be "long." We are primarily interested in learning whether real communities are well described by metrics that expect connections throughout the set, and so understanding

[1]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.

Table I. Roundness Statistics: Ratios of Diameters of Annotated Communities, Parts of Annotated Communities, and Network of Parts of Annotated Communities to Diameters of Random Graphs of the Same Size

	Grad	Ugrad	HS	SC	DM	Amazon	Manu
Ann. Comm. Roundness All sizes	1.34	1.29	1.11	1.17	1.21	1.16	1.23
Ann. Comm. Roundness # nodes $\geq$ 50	1.56	1.47	1.17	1.29	1.50	1.72	1.33
Ann. Comm. Roundness # nodes $\geq$ 75	1.58	1.54	1.13	1.32	1.50	1.84	–
Ann. Comm. Roundness # nodes $\geq$ 100	1.70	1.67	1.15	1.30	1.75	1.91	–
Comm. Parts Roundness	1.07	1.16	1.04	1.06	1.01	1.03	1.02
Comm. Parts Network Roundness	1.02	1.00	1.00	0.99	0.91	1.00	1.00

why a community is not "round" is less important than knowing whether it is or is not "round."

In this section, to estimate a community's roundness value, 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 are unable to generate a connected graph, we discard that community from consideration. This typically occurs when a graph has many nodes and relatively few edges. We then measure the ratio of the community's diameter to the diameter of the random graph.

4.2. 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.[2] 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. We note that all of the annotated communities that we consider are identifying nodes that share some important feature, and thus their structure may not be representative of all true communities (for example, groups of individuals who form a friendship group may not necessarily share any common traits).

The first row of Table I 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 communities 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.

[2]All network features in this chapter were calculated using the NetworkX software package for Python [Hagberg et al. 2008].

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 comparison procedure as before. The fifth row of Table I 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 $n_1, n_2, \ldots, n_k$ in N represents one of the parts $P_1, P_2, \ldots, P_k$ 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 $\frac{T_i}{k}$ 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 five 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 sixth row of Table I. 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 much 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 previous experiments. For each annotated community, community part, or network of parts, we calculate the following features, and compare them to the values 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 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 [Brandes 2001].
—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 [Wasserman and Faust 1994].

Table II. Ratios of Median Edge Betweenness and Transitivity of Annotated Communities, Annotated Community Parts, and Network of Parts to Median Edge Betweenness and Transitivity of Random Graphs of the Same Size

		Grad	Ugrad	HS	SC	DM	Amazon	Manu
Edge Bet.	Ann. Comm. Ratios (size $\geq$ 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
Transitivity	Ann. Comm. Ratios (size $\geq$ 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

Table III. Roundness Statistics: Ratios of Diameters of Annotated Community Parts, Identified through Infomap and Link Communities Algorithms, to Diameters of Random Graphs of the Same Size

	Grad	Ugrad	HS	SC	DM	Amazon	Manu
Comm. Parts Roundness (InfoMap)	1.02	1.14	0.99	1.02	1.04	0.98	1.05
Comm. Parts Roundness (Link Communities)	1.00	1.00	0.99	1.00	1.00	0.99	1.01
Network of Comm. Parts Roundness (InfoMap)	1.02	0.98	1.00	1.02	0.89	1.00	1.00

Table II 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 sets found by greedy modularity optimization are "round." Rather, there are two key points: first, that annotated communities themselves are not well modeled by metrics that are based on expectations of connectivity through the entire community, and second, that annotated communities can be decomposed into smaller parts that are themselves both individually structured and connected to one another in a way that may be correctly captured by such metrics.

Despite this, in order to demonstrate that these results are not simply an artifact of the greedy modularity optimization partitioning algorithm that we chose to use, we apply the InfoMap and Link Communities algorithms to each annotated community and calculate the roundness statistics of the resulting parts. These values are presented in Table III. We see that, as before, these community parts are much rounder than the original annotated communities themselves. Additionally, for the community parts identified through the InfoMap algorithm, we again calculate the roundness statistics of the network of community parts, and obtain similar results. (We do not do this for the Link Communities algorithm, which identifies overlapping communities rather than a network partitioning, because our method of creating the network of parts is not appropriate for overlapping parts).

Working from this intuition, we next present the Node Perception template for finding overlapping communities.

5. NODE PERCEPTION ALGORITHM TEMPLATE

Using the results from the previous section, we now present our Node Perception algorithm template for finding overlapping communities. 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

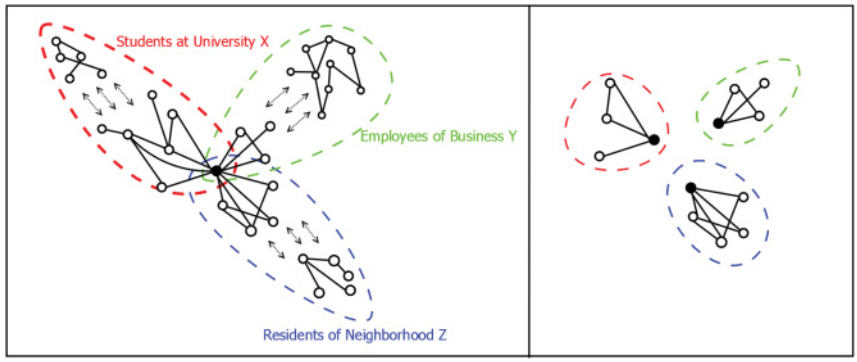

Fig. 1. Creation of subcommunities: Group each node's neighborhood into subcommunities.

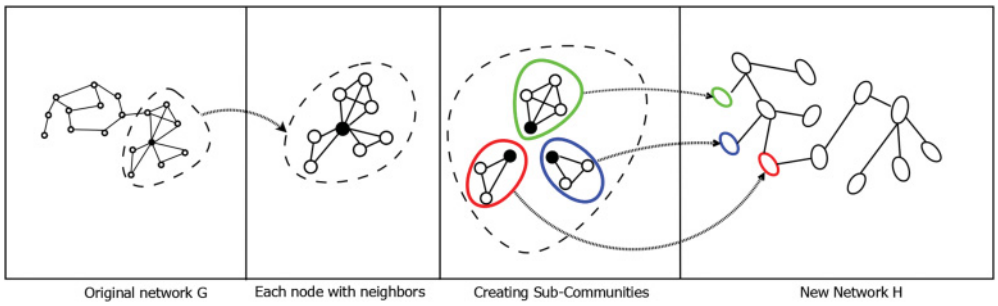

Fig. 2. Algorithm overview: First, identify subcommunities around each node. Second, create a network of subcommunities. Third, group subcommunities into communities.

this section, we give a broad overview of our template, and then discuss the various implementations that we analyze.

5.1. Overview

Our method consists of three parts. First, for every node v in a network G, we separate v's neighbors into subcommunities, 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 subcommunity from the first step. Two nodes in H are connected if their associated subcommunities 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 subcommunities, and each subcommunity into its member nodes from G. Each node can appear in multiple subcommunities, 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 1 and 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 description are intentionally nonspecific. We present precise details of our implementations in the next section.

5.2. Detailed Description

5.2.1. Creation of Subcommunities. Our analysis in Section 4 demonstrated that annotated communities may be well described by a model that joins together small

"subcommunities." However, it is not immediately clear how one might construct these subcommunities. In this article, we create these subcommunities by considering the node neighborhoods of each individual node. Work by Gleich and Seshadhri, demonstrating that node neighborhoods typically have low conductance scores [Gleich and Seshadhri 2012], provides some justification for this method, but other methods are likely possible.

The first step of this method is to identify subcommunities in network G. To do this, we iterate over each node v in network G and group v's neighbors into one or more subcommunities. A node v may belong to many large communities $C_1, C_2, \ldots,$ each containing v and some of v's neighbors, as well as more distant nodes. Each subcommunity $S_{1,v}, S_{2,v}, \ldots$ contains v and those of v's neighbors that belong to $C_1, C_2, \ldots$; that is, the subcommunities correspond to v's local perception of the large communities. For example, in a social network, separate subcommunities may represent the groups of the 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 coworkers).

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 article, we use simple partitioning methods, which are sufficient for good performance on many types of networks, as well as Link Communities, which finds overlapping communities.

The same subcommunity may be created multiple times; for example, when considering subcommunities centered around node a, we may create subcommunity $\{a, b, c\}$. Later, when considering subcommunities around node b, we may again create subcommunity $\{a, b, c\}$. We choose to allow multiple copies of the same subcommunity, but one may 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 subcommunities centering around A, we will place B into only one of A's subcommunities (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 subcommunties 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 subcommunity 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 subcommunities. For example, if G is the Facebook network, then in addition to using a partitioning method to create the subcommunities, groups of people frequently appearing

together in Facebook photographs could be used to create additional subcommunities. If G is a coauthorship network, then an additional subcommunity 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 article (with the exception of a case study in Section 9.3.4); 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.

5.2.2. Creation of Subcommunity Network. In the second step, we create a new network H such that every node in H represents one subcommunity from the first step. Two nodes in H are joined by an edge if the subcommunities 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 subcommunities. One may also require that two subcommunities 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.

5.2.3. Identifying Communities in the Subcommunity Network. Finally, based on the results from Section 4 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 subcommunities represented by the nodes in C. Because each node v can appear in multiple subcommunities 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 subcommunities from C (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 subcommunity in C could be interpreted as related to the strength of v's membership in C.

6. 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 "subcommunities." For example, in Link Communities, "subcommunities" are simply edges from the original network, and Clique Percolation requires subcommunities to be cliques of a fixed size. In contrast, our methods use a community detection method to find subcommunities. This means that subcommunities may have varying sizes and degrees

of connectivity, depending on the structure of the network in that location. Second, other methods typically join subcommunities together by identifying connected components. Node Perception, however, can require more than simple connectedness: Using a community detection algorithm to group the subcommunities ensures a high degree of connectivity between these local groups.

In each implementation described, we perform the first step of identifying subcommunities 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 in the following). To each of these sets, we add v, and thus obtain the subcommunities.

In network H, we create one node representing each subcommunity from the first step. If the same subcommunity 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 subcommunities.[3] 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 subcommunities.

We also attempted to use methods for finding overlapping communities, but this resulted in a large increase in the number of subcommunities, sometimes making network H impractically large. We also experimented with other methods of creating network H, such as setting different similarity thresholds or by weighting edges by similarity, but the experimental results were effectively the same, and so we do not present 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 subcommunities by applying Louvain greedy modularity optimization to each K_v subgraph. Network H is formed as described previously, 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. Modularity maximization algorithms attempt to find "round" communities, with connections throughout the entire set. As we saw in Section 4, 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. Thus, in this implementation, we use modularity maximization both to identify subcommunities as well as to join those subcommunities.

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 subcommunities. As with Mod-Mod, this method finds "round" subcommunities, 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 subcommunities are formed by identifying connected components of each node neighborhood K_v. Although a connected component

[3]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 [Jaccard 1901].

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

is not necessarily "round," this method is likely to be faster than either of the previous two methods. Additionally, because the K_v subgraphs are very small, it may be the case that sophisticated methods such as greedy modularity optimization or Infomap may produce very similar results to much simpler methods, such as this one.

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 [Fortunato and Barthelemy 2006].[4] Because the networks H of subcommunities may be quite large, we also consider other community detection methods.

In the Mod-LC implementation, we again use greedy modularity optimization to identify subcommunities, 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 the process described earlier. 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 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

[4]The resolution limit is an issue with modularity optimization algorithms in general, not just this particular algorithm.

of recursion, only one or two steps were possible for most networks. Indeed, as with Mod-LC, we were again unable to apply this method with even one step of recursion to one network (DM). For the other networks, the results presented were obtained by using one step of recursion.

DEMON/Label Propagation-Subset (LP-Subs): The DEMON algorithm is a specific instance of our Node Perception template, and was created independently by Coscia et al. [2012]. In this algorithm, the initial identification of subcommunities is performed using the Label Propagation algorithm described in Raghavan et al. [2007]. 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 Coscia et al. [2012] and that we consider here, the communities in H consist of those subcommunities C such that there is no other subcommunity D that is a proper superset of C. Because the Label Propagation and subset determination steps are quite fast, this method scales very well.

7. 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, such as conductance or modularity, 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 Section 3.

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 clearly poor communities, such as the set of all people whose names begin with the letter "K."[5] This is not a problem with the approach that we take in this article. Because our goal is to compare our algorithm's performance against that 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) [Ahn et al. 2010] and Backstrom et al. (who used metadata to study the evolution of communities over time) [Backstrom et al. 2006]. Although other evaluation methods are common, they are based on *a priori* assumptions about the structure of communities

[5]Because we identify annotated communities by locating connected components within the subgraph defined by the set of nodes that share some common feature, we know that every annotated community is at least connected, but cannot guarantee meaningful structure beyond this. However, see Abrahao et al. [2012], which analyzes the structure of annotated communities from many of the same datasets used in this article and indicates that the class of annotated communities does have cohesive structure.

and networks, and so can be biased toward particular algorithms. In contrast, using metadata to identify annotated communities allows for an equal, unbiased comparison between many different types of algorithms.

8. 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.

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.

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 Ugrad, 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).

The Infomap algorithm is nondeterministic, and one parameter of the algorithm specifies the number of partition attempts. We are not aware of a method for selecting the optimal number of iterations. Thus, for most of the networks, we perform 10 iterations, as was done in the Infomap documentation. For networks Amazon and DM, which are significantly larger or denser, we only performed one iteration.

For Mod-LC, 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 $0.1, 0.2, \ldots, 0.9$, compute the partition densities obtained by each partitioning, and select the partitioning that obtains the greatest partition density. In experiments on smaller networks, this method proved to give results that were largely indistinguishable from the original method.

All of the algorithms that we consider, including the Node Perception implementations, may at times identify very small communities. OSLOM, Infomap, and greedy modularity optimization may return communities containing as few as one node, Link Communities and the Node Perception methods may identify communities with only two nodes, and Clique Percolation may return communities with only four nodes. Rather than set different thresholds for the minimum size of a "meaningful" community, we simply consider all communities returned by each algorithm. Observe that this decision cannot result in lower recall rates than if we had chosen to set a minimum size threshold; however, it can and does result in lower precision rates for nearly every algorithm.

8.1. 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_A's and report this value in Table IV. This is the "Continuous Recall" score of the algorithm. Although other methods of comparison are certainly

Table IV. Recovery of Annotated Communities (Continuous Recall): Average Jaccard Similarity between Each Annotated Community and the Most Similar Detected Community

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

Table V. Recovery of Annotated Communities (Binary Recall): Fraction of Annotated Communities for Which There is a Detected Community with a Jaccard Similarity of at Least $\frac{1}{3}$

Network	NP: Comps-Mod	NP: IM-Mod	NP: Mod-Mod	NP: Mod-LC	NP: Mod-NP	NP: LP-Subs	LC	CP	OS	IM	Mod
Amazon	0.67	0.53	0.63	0.62	**0.74**	0.61	0.54	0.44	0.58	0.58	0.15
Grad	**0.71**	**0.71**	0.67	**0.71**	**0.71**	**0.71**	0.63	0.54	**0.71**	**0.71**	0.50
Ugrad	0.34	0.39	0.39	0.39	**0.41**	0.37	0.22		0.24	0.22	0.22
HS	0.13	0.16	0.16	0.17	**0.19**	0.11	0.09	0	0.09	0.04	0
SC	0.14	0.19	**0.26**	0.25	0.23	0.14	0.19		0.12	0.05	0
DM	0.25	0.46	**0.50**			0.34	0.43		0.18	0.16	0.05
Manu	0.70	0.70	0.70	0.70	**0.80**	0.70	0.60	0.30	0.60	0.60	0.60

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: for example, 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 V 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_C's (Table VI), or the fraction of such J_C's that are at least $\frac{1}{3}$ (Table VII).

For ease of interpreting results, we have chosen to set a single threshold for calculating binary recall and precision values. Naturally, there is nothing fundamental about the value $\frac{1}{3}$, and we might have chosen any other reasonable threshold instead. The website of this article's first author contains a longer version of this article with an Appendix with box plots containing the distribution of recall and precision values for communities in each network (that is, these plots contain the distribution of J_A and J_C values for each network.) In this version, we summarize the results obtained at different recall and precision thresholds.

Table VI. Recovery of Annotated Communities (Continuous Precision): Average Jaccard Similarity between Each Detected Community and the Most Similar Annotated Community

Network	NP: Comps-Mod	NP: IM-Mod	NP: Mod-Mod	NP: Mod-LC	NP: Mod-NP	NP: LP-Subs	LC	CP	OS	IM	Mod
Amazon	0.09	0.13	0.08	0.12	*0.14*	0.13	0.05	0.05	0.06	**0.16**	0.12
Grad	0.20	0.22	0.14	0.19	*0.29*	0.24	0.07	0.13	0.11	0.30	**0.46**
Ugrad	0.20	0.15	0.12	0.20	*0.33*	0.21	0.04		0.11	0.60	**0.81**
HS	0.02	0.05	0.03	0.05	0.05	***0.06***	0.03	0.04	0.00	0.02	0.02
SC	0.04	0.06	0.05	0.06	0.07	*0.08*	0.03		0.01	0.05	**0.12**
DM	0.01	0.04	0.02			*0.05*	0.01		0.01	0.07	**0.06**
Manu	0.67	0.63	0.42	0.40	0.68	0.65	0.41	0.25	*0.96*	**1.0**	**1.0**

Table VII. Recovery of Annotated Communities (Binary Precision): Fraction of Detected Communities for Which There Is an Annotated Community with a Jaccard Similarity of at Least $\frac{1}{3}$

Network	NP: Comps-Mod	NP: IM-Mod	NP: Mod-Mod	NP: Mod-LC	NP: Mod-NP	NP: LP-Subs	LC	CP	OS	IM	Mod
Amazon	0.060	*0.125*	0.040	0.080	0.109	0.114	0.013	0.020	0.051	**0.175**	0.128
Grad	0.215	0.241	0.067	0.140	*0.376*	0.293	0.020	0.031	0.135	0.333	**0.667**
Ugrad	0.104	0.101	0.031	0.161	*0.506*	0.128	0.001		0.112	0.643	**0.900**
HS	0.001	0.004	0.001	0.003	***0.006***	0.006	0.000	0.000	0.001	0.005	0.000
SC	0.003	0.005	0.004	0.003	0.008	***0.010***	0.001		0.007	0.009	0.000
DM	0.002	0.019	0.008			*0.020*	0.001		0.003	0.081	**0.097**
Manu	***1.000***	0.878	0.657	0.394	0.919	0.930	0.619	0.250	***1.000***	**1.000**	**1.000**

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). For this reason, rather than presenting a simpler evaluation in which precision and recall are combined into a single score (e.g., the F-score), we present the values separately. A user's choice of algorithm ought to be application dependent, and the type of application that demands a high precision may be very different from an application that requires a high recall.

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 VI and VII.

8.2. Results

Our results show that across seven datasets, the Node Perception algorithms generally have significantly higher continuous recall scores than the other methods (Table IV). On each of the networks, as measured by continuous recall, the best performing algorithm was a Node Perception method. When evaluated by binary recall, we again see that the top algorithm in every case is a Node Perception implementation (although on network Grad, several algorithms are in the top position). When calculating the average continuous recall over all networks, we see that IM-Mod and Mod-Mod had an average continuous recall of 0.38, a 15% improvement over Link Communities, which at an average continuous recall of 0.33 was the best non-Node Perception algorithm. Comps-Mod and LP-Subs (DEMON) did not perform as well, but on average, still outperformed every non-Node Perception algorithm. We see qualitatively similar results when evaluating algorithms using binary recall (Table V). 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. In contrast, the Mod-NP implementation produced dramatically better recall scores, but could not be applied to network DM. On five of the six 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 VI and VII). When we examine continuous precision scores, we see that with few exceptions, every Node Perception implementation outperforms every other method for finding overlapping communities. In particular, Mod-NP and LP-Subs perform very well. Each of these methods had the highest continuous precision scores of the algorithms for finding overlapping communities on three of the networks. On network HS, we see that LP-Subs had the highest continuous precision score, even compared to algorithms for partitioning a network. The results for binary precision are somewhat less clear, although we again see that in nearly every case, the best performing algorithm for finding overlapping communities was a Node Perception method. Averaged over all seven networks (or, in the case of Mod-LC and Mod-NP, six networks), four of the six Node Perception implementations outperform the other methods for finding overlapping communities. Again, Mod-NP in particular has very high continuous precision scores.[6]

As noted before, one can easily make an algorithm with very high recall and very low precision (or vice versa). Thus, it is critical to note here that Node Perception achieves its high recall scores while, in most cases, *matching* or *exceeding* the precision scores of other algorithms to find overlapping communities.

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.

When considering different thresholds for binary recall and precision, we see similar results. Notably, the Mod-NP implementation again performs very well on most datasets. Full details of these results are available in a longer version of this article on the first author's website.

As mentioned earlier, when analyzing algorithm output we chose not to set a threshold for minimum community size. When we experimented with different thresholds, we observed virtually identical results for recall: even with such thresholds, the Node Perception implementations outperformed all of the other algorithms. In the precision results presented, we saw that Node Perception performed roughly equally to the other methods for finding overlapping communities. When we calculated precision using only communities larger than the threshold, we saw similar results on average, but with more variance over different networks and implementations.

8.3. Discussion

The main implication of these results is that the general principle of joining together small subcommunities of varying size and structure is of far more importance than the specific implementation used. These results confirm the analysis in Section 4, which showed that annotated communities are formed of well-connected sets that are

[6]The results presented were obtained by setting no minimum size threshold for detected communities. When we experimented with different minimum size thresholds and calculated recall and precision scores, the algorithms were ranked in roughly the same order as with no threshold, with the exception of the OSLOM algorithm, which is intended to find overlapping communities. Interestingly, when we set a size threshold and calculated precision scores, we observed that OSLOM behaved like a partitioning algorithm: it had excellent precision scores, but almost every node appeared in only one community. In contrast, with the other algorithms for finding overlapping communities, almost every node appears in multiple communities.

well connected to one another. While the resource-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 the features of the network under consideration.

9. SCALABILITY AND SUITABILITY

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

9.1. 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 15min, 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 subcommunities. 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 subcommunity 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 12h 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 subcommunities, requiring a greater amount of overlap between subcommunities, or using a simple clustering scheme to identify communities in H, rather than a slower, more resource intensive method such as Link Communities.

9.2. Network Suitability

The Node Perception algorithm as we have presented it is highly dependent on network transitivity. To create the subcommunities, we consider the neighbors of a node and their connections to one another. Creating subcommunities 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 three-cycles (transitivity) the network may instead have many four-cycles. In such a case, we might modify the method used to identify the subcommunities. For example, if the network has many four-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.

9.3. 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 case studies.

9.3.1. 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 subcommunities, 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 subcommunities, 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 0.14), and a binary precision score of 0.272 (compared with 0.066). 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 subcommunities 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.

9.3.2. Network Grad. In this case study, we again consider network Grad, but turn to a different problem. Suppose one wishes to create committees from a group of people; for example, one committee of grad students might be responsible for welcoming incoming

students, and another committee might be responsible for organizing talks. The person in charge of creating these committees has a fixed number of committees in mind, and wants to find groups that are overlapping in order to facilitate the flow of ideas between different committees. Although the data that we consider is for graduate students, one can easily perform the same analysis on a network of faculty members or of employees in some business. Suppose that the organizer wishes to identify 50 overlapping communities.

To accomplish this task, we can use the clustering algorithm METIS, which finds a specified number of clusters, to produce the Mod-METIS implementation [Karypis and Kumar 1998]. Unlike the other implementations that we consider in this article, this implementation allows us to find a specified number of overlapping communities. When we apply this algorithm to network Grad to find 50 communities, we achieve a continuous recall of 0.42, a binary recall of 0.71, a continuous precision of 0.35, and a binary precision of 0.54. Mod-METIS's continuous recall is lower than that of the other Node Perception implementations, though its binary recall is the same. Interestingly, it has much higher precision scores. We believe that this is because, as with the greedy modularity optimization and Infomap community detection algorithm, this implementation finds a fairly small number of communities. Unlike those algorithms, however, this implementation finds overlapping communities and allows the user to specify the number of communities.

9.3.3. 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 [Clauset et al. 2009]. The creation of subcommunities is thus usually fast, but sometimes slow. We demonstrated earlier that if we created subcommunities 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 subcommunities using greedy modularity optimization. This process produces results that are 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.

9.3.4. 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 [Stumpf et al. 2008; Amaral 2008]. 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 subcommunities 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 subcommunity containing n and all of its neighbors from C. If n is known to

play multiple genetic functions, then there may be multiple such subcommunities, and we add all of them to the list of subcommunities.

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 genetic functions, and use this information to produce the additional subcommunities. 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 0.028 to 0.032, and the binary precision from 0.001 to 0.002. 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, continuous precision from 0.048 to 0.051, and binary precision from 0.004 to 0.006.

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.

10. 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 copurchasing 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 intuition, we developed the Node Perception algorithm template for finding overlapping communities in networks. In this template, we first identify subcommunities 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 subcommunity, and two nodes are linked if their associated subcommunities overlap by at least some threshold amount. Finally, we identify communities in this new network, and for every such community, merge together the associated subcommunities 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; these results strongly suggest 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 [Saramäki et al. 2007], degree distribution, assortativity [Newman 2002], and so on. Although even simple implementations of Node Perception were quite successful, such modifications may further improve performance.

Additionally, we are interested in identifying 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 "subcommunity' as we have described in this article? 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.

REFERENCES

Bruno Abrahao, Sucheta Soundarajan, John Hopcroft, and Robert Kleinberg. 2012. On the separability of structural classes of communities. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 624–632.

Balázs Adamcsek, Gergely Palla, Illés J. Farkas, Imre Derényi, and Tamás Vicsek. 2006. CFinder: Locating cliques and overlapping modules in biological networks. *Bioinformatics* 22, 8 (2006), 1021–1023.

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

Luis A. Amaral. 2008. A truer measure of our ignorance. *Proc. Natl. Acad. Sci.* 105, 19 (2008), 6795–6796.

Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. 2006. Group formation in large social networks: Membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* ACM, 44–54.

Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *J. Stat. Mech.* 2008 (2008), P10008.

Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. *J. Math. Soc.* 25, 2 (2001), 163–177.

Aaron Clauset, Cosma Shalizi, and Mark Newman. 2009. Power-law distributions in empirical data. *SIAM Rev.* 51, 4 (2009), 661–703.

Michele Coscia, Giulio Rossetti, Fosca Gianotti, and Dino Pedreschi. 2012. DEMON: A local-first discovery method for overlapping communities. *Proceedings of the 18th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2012). ACM, 615–623.

Robert L. Cross, Andrew Parker, and Rob Cross. 2004. *The Hidden Power of Social Networks: Understanding How Work Really Gets Done in Organizations*. Harvard Business School Press.

Paul Erdős and Alfred Rényi. 1959. On random graphs I. *Publ. Math. Debrecen* 6 (1959), 290–297.

Santo Fortunato. 2010. Community detection in graphs. *Phys. Rep.* 486, 3–5 (2010), 75–174.

Santo Fortunato and Marc Barthelemy. 2006. Resolution limit in community detection. *Proc. oNatl. Acad. Sci.* 104, 1 (2006), 36–41.

Adrien Friggeri, Guillaume Chelieu, and Eric Fleury. 2011. *Egomunities, Exploring Socially Cohesive Person-Based Communities*. INRIA Research Report RR-7535.

Michelle Girvan and Mark Newman. 2002. Community structure in social and biological networks. In *Proc. Natl. Acade. Sci.* 99, 12 (2002), 7821–7826.

David Gleich and C. Seshadhri. 2012. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *Proceedings of the 18th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2012). ACM, 597–605.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference* (2008). 11–15.

Paul Jaccard. 1901. Etude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bull. Soc. Vaudoise Sci. Nat.* 37, 140 (1901), 241–272.

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

George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* 20, 1 (Aug. 1998), 359–392.

Andrea Lancichinetti and Santo Fortunato. 2009. Community detection algorithms: A comparative analysis. *Phys. Rev. E* 80 (2009), 056117.

Andrea Lancichinetti, Filippo Radicchi, José Ramasco, and Santo Fortunato. 2011. Finding statistically significant communities in networks. *PLoS ONE* 6, 4 (2011), e18961.

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

Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. 2005. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *Proceedings of the 9th European Conference on Principles and Practices of Knowledge Discovery in Databases* (2005). Springer-Verlag, 133–145.

Alan Mislove, Bimal Viswanath, Krishna Gummadi, and Peter Druschel. 2010. You are who you know: Inferring user profiles in online social networks. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining*. ACM, 251–260.

Mark Newman. 2002. Assortative mixing in networks. *Phys. Rev. Lett.* 89 (2002), 208701.

Michael Newman. 2006. Modularity and community structure in networks. *Proc. Natl. Acad. Sci.* 103, 23 (2006), 8577–8582.

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

Daniel Park, Rohit Singh, Michael Baym, Chung-Shou Liao, and Bonnie Berger. 2011. IsoBase: A database of functionally related proteins across PPI networks. *Nucleic Acids Rese.* 39, D295–D300.

Usha Raghavan, Albert Reka, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E* 76, 3 (2007), 036106.

Martin Rosvall and Carl Bergstrom. 2008. Maps of random walks on complex networks reveal community structure. *Proc. Nat. Acad. Sci.* 105, 4 (2008), 1118–1123.

Jari Saramäki, Mikko Kivelä, Jukka-Pekka Onnela, Kimmo Kaski, and János Kertész. 2007. Generalizations of the clustering coefficient to weighted complex networks. *Phys. Rev. E* 75, 2 (2007), 027105.

Rohit Singh, Jinbo Xu, and Bonnier Berger. 2008. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proc. Natl. Acad. Sci.* 105, 35 (2008), 12763–12768.

Michael Stumpf, Thomas Thorne, Eric de Silva, Ronald Stewart, Hyeong Jun J. An, Michael Lappe, and Carsten Wiuf. 2008. Estimating the size of the human interactome. *Proc. Natl. Acad. Sci.* 105, 19 (2008), 6959–6964.

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

Jaewon Yang and Jure Leskovec. 2012. Defining and evaluating network communities based on ground-truth. In *Proceedings of the IEEE International Conference on Data Mining*. IEEE, 745–754.

Received October 2012; revised June 2014; accepted September 2014