Community Structure and Detection in Complex Networks: A Survey

Guoliang Liu

CSC8350 Literature Review (Due day: 25/04/2012)
Instructor: Dr. Sushil Prasad
Abstract

Community structure is common in various real-world networks. Methods or algorithms for detecting such communities in complex networks have attracted great attention in recent years. From a broad perspective of understanding community detection, the study of community structure in networks has a long history. It is closely related to the ideas of graph partitioning in graph theory and computer science, and hierarchical clustering in sociology. Detecting communities is of great importance in sociology, biology and computer science, disciplines where systems are often represented as graphs. This problem is very hard and not yet satisfactorily solved since it has been proved as a NP-hard problem. In this survey, we try to exploit the community detection from the basic issues related such as definitions, a classification of methods and algorithms to the tested benchmarks and real-world applications used in existed algorithms and methods.

I. Introduction

Community detection is very similar with partition in graph theory. In our real-world life, Biological, social, technological, semantic and information networks can all be studied as graphs. As a result, graph analysis has become very important to understand the features of these networks. One of the features in all this networks is called community or cluster. In recent times, the computer revolution has provided scholars with a huge amount of data and computational resources to process and analyze these data. The size of real networks one can potentially handle has also grown considerably, reaching millions or even billions of vertices. The need to deal with such a large number of units has produced a deep change in the way graphs are approached (Albert and BarabSasi, 2002; Barrat et al., 2008; Boccaletti et al., 2006; Mendes and Dorogovtsev, 2003; Newman, 2003; Pastor-Satorras and Vespignani, 2004).

In a random graph, the distribution of edges among the vertices is highly homogeneous. For instance, the distribution of the number of neighbors of a vertex, or degree, is binomial, so most vertices have equal or similar degree. Real networks are not random graphs, as they display big inhomogeneities, revealing a high level of order and organization. The degree distribution is broad, with a tail that often follows a power law: therefore, many vertices with low degree coexist with some vertices with large degree. Furthermore, the distribution of edges is not only globally, but also locally inhomogeneous, with high concentrations of edges within special groups of vertices, and low concentrations between these groups. This feature of real networks is called community structure (Girvan and Newman, 2002), there is an
example of this community structure (see Fig 1). Community structure is one property of real-

world networks and community detection also has important applications. Clustering Web

clients who have similar interests and are geographically near to each other may improve the

performance of services provided on the World Wide Web, in that each cluster of clients could

be served by a dedicated mirror server (Krishnamurthy and Wang, 2000). Clusters of large

graphs can be used to create data structures in order to efficiently store the graph data and to

handle navigational queries, like path searches (Agrawal and Jagadish, 1994; Wu et al., 2004).

In Mobile social networks, for better message dissemination, the message can be forwarded
to certain influential nodes in a community which have large influence on other nodes in the

network (D. Kempe and J. Kleinberg, 2005). Identifying graph communities is a popular topic

in computer science, too. In parallel computing, for instance, it is crucial to know what is the

best way to allocate tasks to processors so as to minimize the communications between them

and enable a rapid performance of the calculation. This can be accomplished by splitting

the computer cluster into groups with roughly the same number of processors, such that the

number of physical connections between processors of different groups is minimal.

This manuscript is organized as below. In the section II, we will first introduce the complex

networks and the properties. Community structure as one property is exploited in this survey

and some other properties need to be considered in some circumstances. First, the definition

of community structure is given and the the basic idea of finding community detection is

introduced and at the last part of this section, the computational complexity is talked about. In

the section III, a classification of all kinds of algorithms is given, which includes four classes,
which are agglomerative, divisive, constructive algorithms and optimization approach. In the section IV, Benchmarks to test the detection algorithms are talked about. In the last section V, we conclude the survey.

A. Complex Networks Features

Complex networks are the Systems taking the form of networks (also called graphs in much of the mathematical literature) abound in the world. Examples include the Internet, the World Wide Web, social networks of acquaintance or other connections between individuals, organizational networks and networks of business relations between companies, neural networks, metabolic networks, food webs, distribution networks such as blood vessels or postal delivery routes, networks of citations between papers, and many others. Before we study the network or graph structure, the properties of the networks should be exploited first. Newman summarized the properties of networks in 2003 that contains the small-world effect, transitivity or clustering, degree distributions, network resilience, mixing patterns, degree correlations, community structure, network navigation and other network properties. Community structure as one property enables community detection useful and other properties should be also considered in community detection. E.g., degree distribution is considered when finding community structure (Thang N.Dinh and My T.Thai, 2011). Self-similarity property is used in a fast unfolding community detection algorithm (Vincent D.Blondel, Jean-Loup Guillaume, Renaud Lambiotte and Etienne Lefebvre, 2008). This property is proved to exist in complex networks (Chaoming Song, Shlomo Havlin and Hernan A.Makse, 2005).

B. Issues in Community Detection

The problem of graph clustering or partition is actually not well defined. The concepts of community and partition are not rigorously defined. Indeed, some ambiguities are hidden and there are often many equally legitimate ways of resolving them. Therefore, it is not surprising that there are plenty of recipes in the literature and that people do not even try to ground the problem on shared definitions.

Graph has the property of community structure only if graphs are sparse or weighted when it is dense, i.e. if the number of edges m is of the order of the number of nodes n of the graph. If m is much larger than n, the distribution of edges among the nodes is too homogeneous for communities to make sense.

In this section, we try to exploit the fundamental concepts of community detection. Based on the definitions, partition metrics will be reviewed about and at last computational complexity for the algorithms is taken into discussion.
1) **Community Structure Definition:** The first problem in graph clustering is to look for a quantitative definition of community. No definition is universally accepted. As a matter of fact, the definition often depends on the specific system at hand and/or application one has in mind. From intuition and the examples of Section II we get the notion that there must be more edges inside the community than edges linking vertices of the community with the rest of the graph. This is the reference guideline at the basis of most community definitions. But many alternative recipes are compatible with it. Moreover, in most cases, communities are algorithmically defined, i.e. they are just the final product of the algorithm, without a precise a priori definition.

We define the intra-cluster density \( \delta_{\text{int}}(C) \) of the subgraph \( C \) as the ratio between the number of internal edges of \( C \) and the number of all possible internal edges, i.e.

\[
\delta_{\text{int}}(C) = \frac{\# \text{internal edges of } C}{n_c(n_c-1)/2}
\]

Similarly, the inter-cluster density \( \delta_{\text{ext}}(C) \) is the ratio between the number of edges running from the vertices of \( C \) to the rest of the graph and the maximum number of inter-cluster edges possible, i.e.

\[
\delta_{\text{ext}}(C) = \frac{\# \text{inter-community edges of } C}{n_c(n-c)}/n_c(n-c)
\]

For \( C \) to be a community, we expect \( \delta_{\text{int}}(C) \) to be appreciably larger than the average link density \( \delta(G) \) of \( G \), which is given by the ratio between the number of edges of \( G \) and the maximum number of possible edges \( n(n-1)/2 \). On the other hand, \( \delta_{\text{ext}}(C) \) has to be much smaller than \( \delta(G) \). Searching for the best tradeoff between a large \( \delta_{\text{int}}(C) \) and a small \( \delta_{\text{ext}}(C) \) is implicitly or explicitly the goal of most clustering algorithms. A simple way to do that is, e.g., maximizing the sum of the differences \( \delta_{\text{int}}(C) - \delta_{\text{ext}}(C) \) over all clusters of the partition (Mancoridis et al., 1998).

2) **Community Partition:** 1. Partition

A partition is a division of a graph in clusters, such that each vertex belongs to one cluster. In real systems vertices may be shared among different communities. A division of a graph into overlapping communities is called cover.

Partitions can be hierarchically ordered, when the graph has different scales. In this case, clusters display in turn community structure, with smaller communities inside, which may contain smaller communities and so on. A natural way to represent the hierarchical structure is to draw a hierarchical tree like the one illustrated in Fig.2. Based on this hierarchical structure, two class of methods can be developed using approximation algorithms including agglomerative and divisive methods, both of which will be talked about in section 2. At the same time, there are some other methods and algorithms which dose not obey the
hierarchical structure and hence can not be classified in both of the two methods are classified to constructive methods.

![Hierarchical tree of community structure](image)

**Fig. 2. Hierarchical tree of community structure (From Newman and Girvan, 2004)**

2. Quality metric: modularity

Reliable algorithms are supposed to identify good partitions. But what is a good clustering? In order to distinguish between "good" and "bad" clusterings, it would be useful to require that partitions satisfy a set of basic properties, intuitive and easy to agree upon. In the wider context of data clustering, this issue has been studied by Jon Kleinberg (Kleinberg, 2002), who has proved an important impossibility theorem. Given a set S of points, a distance function d is defined, which is positive definite and symmetric (the triangular inequality is not explicitly required). One wishes to find a clustering f based on the distances between the points. Kleinberg showed that no clustering satisfies at the same time the three following properties:

1. Scale-invariance: given a constant $\alpha$, multiplying any distance function d by $\alpha$ yields the same clustering.

2. Richness: any possible partition of the given point set can be recovered if one chooses a suitable distance function d.

3. Consistency: given a partition, any modification of the distance function that does not decrease the distance between points of different clusters and that does not increase the distance between points of the same cluster, yields the same clustering.

The most popular quality function is the modularity of Newman and Girvan (Newman and Girvan, 2004). It is based on the idea that a random graph is not expected to have a cluster structure, so the possible existence of clusters is revealed by the comparison between the actual density of edges in a subgraph and the density one would expect to have in the
subgraph if the vertices of the graph were attached regardless of community structure. This expected edge density depends on the chosen null model, i.e., a copy of the original graph keeping some of its structural properties but without community structure. Modularity can then be written as follows (Girman and Newman, 2004)

$$Q = \frac{1}{2m} \sum_{ij} \left( A_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j)$$

where the sum runs over all pairs of vertices, A is the adjacency matrix, m the total number of edges of the graph, and $P_{ij}$ represents the expected number of edges between vertices i and j in the null model. The $\delta$-function yields one if vertices i and j are in the same community ($C_i = C_j$), zero otherwise.

From the understanding of this formula, indeed, 

$$Q = \text{(fraction of edges within communities)} - \text{(expected fraction of such edges)}$$

Large positive values of the modularity indicate when a statistically surprising fraction of the edges in a network fall within the chosen communities, it tells us when there are more edges within communities than we would expect on the basis of chance.

Modularity has been employed as quality function in many algorithms, like some of the divisive algorithms. In addition, modularity optimization is itself a popular method for community detection. Modularity also allows to assess the stability of partitions (Massen and Doye, 2006), it can be used to design layouts for graph visualization (Noack, 2009) and to perform a sort of re-normalization of a graph, by transforming a graph into a smaller one with the same community structure (Arenas et al., 2007), moreover, it is also used in an adaptive clustering algorithm (zhengqing Ye, Songnian Hu, and Jun Yu, 2008), recently, there is a work published in Social com studying community structure in scale-free networks targeting maximizing modularity (Thang N, Dinh and My T. Thai, 2011).

3) Computational complexity: The computational complexity of an algorithm is the estimate of the amount of resources required by the algorithm to perform a task. This involves both the number of computation steps needed and the number of memory units that need to be simultaneously allocated to run the computation. In the case of a graph, the size is typically indicated by the number of vertices n and/or the number of edges m. The computational complexity of an algorithm cannot always be calculated. In fact, sometimes this is a very hard task, or even impossible. In these cases, it is however important to have at least an estimate of the worst-case complexity of the algorithm.

In the graph theory, the notation $O(n^\alpha m^\beta)$ indicates that the computer time grows as a power of both the number of vertices and edges, with exponents $\alpha$ and $\beta$, respectively. In
real-world networks, however, e. g. Samples of the Web graph, with millions of vertices and billions of edges, cannot be tackled by algorithms whose running time grows faster than $O(n)$ or $O(m)$. In this condition, NP-problem appears and need to be tackled.

Most of the clustering algorithms or problems related to clustering are NP-hard. In this case, it is pointless to use exact algorithms, which could be applied only to very small systems. Moreover, even if an algorithm has a polynomial complexity, it may still be too slow to tackle large systems of interest. In all such cases it is common to use approximation algorithms, i. e. methods that do not deliver an exact solution to the problem at hand, but only an approximate solution, with the advantage of a lower complexity. In any case, one should give provable bounds on the goodness of the approximate solution delivered by the algorithm with respect to the optimal solution. In many cases it is not possible to approximate the solution within any constant, as the goodness of the approximation strongly depends on the specific problem at study.

II. Heuristic Methods Classification

In section I, we have already introduced the NP-hard problems, and hence it is common to use approximation algorithms. In this section, we will talk about heuristic methods at the goodness of different classifications. At first, we first introduce agglomerative class, which is very common in the early years of exploiting community structure. Since the basic idea of agglomerative methods comes from the hierarchical structure of community structure (figure 2), Another classification of algorithms are developed with the reverse sequence respect to the agglomerative ones. Except for these two classifications of algorithms, there is still another classification of methods called constructive methods also derived from hierarchical structure. At the end, there are still some other methods not based on the hierarchical structure but also worth mentioning.

A. Agglomerative methods

The agglomerative methods use the up-down way on hierarchical structure of community and is based on modularity. When the hierarchical structure was proposed, at the same time, a basic agglomerative method( Newman, 2004) was designed and became a foundation work of the class of agglomerative methods. In this section, first the foundation work mentioned above will be introduced first and then review several representative works based on the first one. As to other agglomerative methods, they will be concisely reviewed and cited.
The first agglomerative algorithms of Newman: The algorithm is based on the idea of modularity. Given any network, the GN community structure algorithm always produces some division of the vertices into communities, regardless of whether the network has any natural such division. To test whether a particular division is meaningful we define a quality function or modularity Q as introduced in section I.2.2.

If a high value of Q represents a good community division, simply optimizing Q over all possible divisions to find the best one should be a way to the goal, we can avoid the iterative removal of edges and cut straight to the chase. The problem is that true optimization of Q is very costly. The number of ways to divide n vertices into g non-empty groups is given by the Stirling number of the second kind \( S_n^{(g)} \), and hence the number of distinct community divisions \( \sum_{g=1}^{n} S_n^{(g)} \). This sum is not known in closed form, but we observe that \( S_n^{(1)} + S_n^{(2)} = 2^{n-1} \) for all \( n > 1 \), so that the sum must increase at least exponentially in n. To carry out an exhaustive search of all possible divisions for the optimal value of Q would therefore take at least an exponential amount of time, and is in practice infeasible for systems larger than twenty or thirty vertices. Various approximate optimization methods are available: simulated annealing, genetic algorithms, and so forth. Here Newman et al, consider a scheme based on a standard greedy optimization algorithm, which appears to perform well.

Newman’s algorithm starts with a state in which each vertex is the sole member of one of n communities, they repeatedly join communities together in pairs, choosing at each step the join that results in the greatest increase (or smallest decrease) in Q. The progress of the algorithm can be represented as a dendrogram, a tree that shows the order of the joins (see Fig.3, for an example). Cuts through this dendrogram at different levels give divisions of the network into larger or smaller numbers of communities and, as with the GN algorithm, we can select the best cut by looking for the maximal value of Q. Since the joining of a pair of communities between which there are no edges at all can never result in an increase in Q, we need only consider those pairs between which there are edges, of which there will at any time be at most \( m \), where \( m \) is again the number of edges in the graph. The change in Q upon joining two communities is given by \( \Delta Q = 2(e_{ij} - a_i a_j) \), which can clearly be calculated in constant time. Following a join, some of the matrix elements \( e_{ij} \) must be updated by adding together the rows and columns corresponding to the joined communities, which takes worst case time \( O(n) \). Thus each step of the algorithm takes worst-case time \( O(m + n) \). There are a maximum of \( n - 1 \) join operations necessary to construct the complete dendrogram and hence the entire algorithm runs in time \( O((m+n)n) \), or \( O(n^2) \) on a sparse
graph. The algorithm has the added advantage of calculating the value of $Q$ as it goes along, making it especially simple to find the optimal community structure. It is worth noting that the GN algorithm can be generalized trivially to weighted networks in which each edge has a numeric strength associated with it, by making the initial values of the matrix elements $e_{ij}$ equal to those strengths, rather than just zero or one; otherwise the algorithm is as above and has the same running time.

2) A modification of Newman algorithm based on self-similarity of complex networks: Modularity has been used to compare the quality of the partitions obtained by different methods, but also as an objective function to optimize. Unfortunately, exact modularity optimization is a problem that is computationally hard and so approximation algorithms are necessary when dealing with large networks. The fastest approximation algorithm for optimizing modularity on large networks was proposed by Clauset, Newman et al., 2004. That method consists in recurrently merging communities that optimize the production of modularity. Unfortunately, this greedy algorithm may produce values of modularity that are significantly lower than what can be found by using. Moreover, the method proposed by Clauset, Newman et al., 2004 has a tendency to produce super-communities that contain a large fraction of the nodes, even on synthetic networks that have no significant community structure. This artefact also has the disadvantage to slow down the algorithm considerably and makes it inapplicable to networks of more than a million nodes. This undesired effect has been circumvented by introducing tricks in order to balance the size of the communities being merged, thereby speeding up the running time and making it possible to deal with
networks that have a few million nodes.

The algorithm is divided in two phases that are repeated iteratively. Assume that we start with a weighted network of N nodes. First, we assign a different community to each node of the network. So, in this initial partition there are as many communities as there are nodes. Then, for each node i we consider the neighbors j of i and we evaluate the gain of modularity that would take place by removing i from its community and by placing it in the community of j. The node i is then placed in the community for which this gain is maximum (in case of a tie we use a breaking rule), but only if this gain is positive. If no positive gain is possible, i stays in its original community. This process is applied repeatedly and sequentially for all nodes until no further improvement can be achieved and the first phase is then complete. Let us insist on the fact that a node may be, and often is, considered several times. This first phase stops when a local maxima of the modularity is attained, i.e., when no individual move can improve the modularity. One should also note that the output of the algorithm depends on the order in which the nodes are considered. Preliminary results on several test cases seem to indicate that the ordering of the nodes does not have a significant influence on the modularity that is obtained. However the ordering can influence the computation time. The problem of choosing an order is thus worth studying since it could give good heuristics to enhance the computation time. The second phase of the algorithm consists in building a new network whose nodes are now the communities found during the first phase. To do so, the weights of the links between the new nodes are given by the sum of the weight of the links between nodes in the corresponding two communities. Links between nodes of the same community lead to self-loops for this community in the new network. Once this second phase is completed, it is then possible to reapply the first phase of the algorithm to the resulting weighted network and to iterate. Let us denote by pass a combination of these two phases. By construction, the number of meta-communities decreases at each pass, and as a consequence most of the computing time is used in the first pass. The passes are iterated (see Fig.4) until there are no more changes and a maximum of modularity is attained. The algorithm is reminiscent of the self-similar nature of complex networks and naturally incorporates a notion of hierarchy, as communities of communities are built during the process. The height of the hierarchy that is constructed is determined by the number of passes and is generally a small number. This simple algorithm has several advantages. First, its steps are intuitive and easy to implement, and the outcome is unsupervised. Moreover, the algorithm is extremely fast.
Fig. 4. Visualization of the steps of the fast unfolding algorithm. Each pass is made of two phases: one where modularity is optimized by allowing only local changes of communities; one where the found communities are aggregated in order to build a new network of communities. The passes are repeated iteratively until no increase of modularity is possible. (From Vincent D. Blondel et al, 2009)

B. Divisive methods

A simple way to identify communities in a graph is to detect the edges that connect vertices of different communities and remove them, so that the clusters get disconnected from each other. This is the philosophy of divisive algorithms. The crucial point is to find a property of intercommunity edges that could allow for their identification.

1) The first divisive algorithm of Girvan and Newman: The most popular algorithm is that proposed by Girvan and Newman (Newman and Girvan, 2004). The method is historically important, because it marked the beginning of a new era in the field of community detection. Edges are selected according to the values of measures of edge centrality, estimating the importance of edges according to some property or process running on the graph. The steps of the algorithm are:

1. Computation of the centrality for all edges;
2. Removal of edge with largest centrality: in case of ties with other edges, one of them is picked at random;
3. Recalculation of centralities on the running graph;
4. Iteration of the cycle from step 2.

Girvan and Newman focused on the concept of between-ness, which is a variable expressing the frequency of the participation of edges to a process. They considered three alternative
definitions: geodesic edge betweenness, random-walk edge betweenness and current-flow edge betweenness. In the following we shall refer to them as edge betweenness, random-walk betweenness and current-flow betweenness, respectively.

First, let us review the geodesic edge betweenness, which is also called shortest-path betweenness. At first sight, it appears that calculating the edge betweenness measure based on geodesic paths for all edges will take \( O(mn^2) \) operations on a graph with \( m \) edges and \( n \) vertices: calculating the shortest path between a particular pair of vertices can be done using breadth-first search in time \( O(m) \), and there are \( O(n^2) \) vertex pairs. However new algorithms have been proposed by Newman and independently by Brandes (Brandes, 2011) that can perform the calculation faster than this, finding all betweennesses in \( O(mn) \) time. Both Newman and Brandes gave algorithms for the standard Freeman vertex betweenness, but it is trivial to adapt their algorithms for edge betweenness. Breadth-first search (Brandes, 2001; Newman and Girvan, 2004; Zhou et al., 2006) can find shortest paths from a single vertex \( s \) to all others in time \( O(m) \).

In the context of information spreading, one could imagine that signals ow across random rather than geodesic paths. In this case the betweenness of an edge is given by the frequency of the passages across the edge of a random walker running on the graph (random-walk betweenness). A random walker moving from a vertex follows each adjacent edge with equal probability. A pair of vertices is chosen at random, \( s \) and \( t \). The walker starts at \( s \) and keeps moving until it hits \( t \), where it stops. One computes the probability that each edge was crossed by the walker, and averages over all possible choices for the vertices \( s \) and \( t \). It is meaningful to compute the net crossing probability, which is proportional to the number of times the walk crossed the edge in one direction. In this way one neglects back and forth passages that are accidents of the random walk and tell nothing about the centrality of the edge. Calculation of random-walk betweenness requires the inversion of an \( n \times n \) matrix (once), followed by obtaining and averaging the flows for all pairs of nodes. The first task requires a time \( O(n^3) \), the second \( O(mn^2) \), for a total complexity \( O((m+n)n^2) \), or \( O(n^3) \) for a sparse matrix. The complete calculation requires a time \( O(n^3) \) on a sparse graph.

Current-flow betweenness is defined by considering the graph a resistor network, with edges having unit resistance. If a voltage difference is applied between any two vertices, each edge carries some amount of current, that can be calculated by solving Kirchoff’s equations. The procedure is repeated for all possible vertex pairs: the current-flow betweenness of an edge is the average value of the current carried by the edge. It is possible to show that this measure is
equivalent to random-walk betweenness, as the voltage differences and the random walks net flows across the edges satisfy the same equations (Newman, 2005). Therefore, the calculation of current-flow betweenness has the same complexity $O((m + n)n^2)$, or $O(n^3)$ for a sparse graph.

Calculating edge betweenness is much faster than current-flow or random walk betweenness ($O(n^2)$ versus $O(n^3)$ on sparse graph). In addition, in practical applications the Girvan-Newman algorithm with edge betweenness gives better results than adopting the other centrality measures (Newman and Girvan, 2004). Numerical studies show that the recalculation step 3 of Girvan-Newman algorithm is essential to detect meaningful communities. This introduces an additional factor $m$ in the running time of the algorithm: consequently, the edge betweenness version scales as $O(m^2n)$, or $O(n^3)$ on a sparse graph. On graphs with strong community structure, that quickly break into communities, the recalculation step needs to be performed only within the connected component including the last removed edge (or the two components bridged by it if the removal of the edge splits a subgraph), as the edge betweenness of all other edges remains the same. This can help saving some computer time, although it is impossible to give estimates of the gain since it depends on the specific graph at hand. Nevertheless, the algorithm is quite slow, and applicable to sparse graphs with up to $n \sim 10000$ vertices, with current computational resources. In the original version of Girvan-Newman’s algorithm (Girvan and Newman, 2002), the authors had to deal with the whole hierarchy of partitions, as they had no procedure to say which partition is the best.

The algorithm of Girvan and Newman is unable to find overlapping communities, as each vertex is assigned to a single cluster. Pinney and Westhead have proposed a modification of the algorithm in which vertices can be split between communities (Pinney and Westhead, 2006). To do that, they also compute the betweenness of all vertices of the graph. Unfortunately the values of edge and site betweenness cannot be simply compared, due to their different normalization, but the authors remarked that the two endvertices of an inter-cluster edge should have similar betweenness values, as the shortest paths crossing one of them are likely to reach the other one as well through the edge.

C. Other Divisive methods

An alternative measure of centrality for edges is information centrality. It is based on the concept of efficiency (Latora and Marchiori, 2001), which estimates how easily information travels on a graph according to the length of shortest paths between vertices. The efficiency of a network is defined as the average of the inverse distances between all pairs of vertices.
If the vertices are close to each other, the efficiency is high. The information centrality of an edge is the relative variation of the efficiency of the graph if the edge is removed. In the algorithm by Fortunato et al. (Fortunato et al., 2004), edges are removed according to decreasing values of information centrality. The method is analogous to that of Girvan and Newman. Computing the information centrality of an edge requires the calculation of the distances between all pairs of vertices, which can be done with breadth-first-search in a time $O(mn)$. So, in order to compute the information centrality of all edges one requires a time $O(m^2n)$. At this point one removes the edge with the largest value of information centrality and recalculates the information centrality of all remaining edges with respect to the running graph. Since the procedure is iterated until there are no more edges in the network, the final complexity is $O(m^3n)$, or $O(n^4)$ on a sparse graph. The partition with the largest value of modularity is chosen as most representative of the community structure of the graph. The method is much slower than the algorithm of Girvan and Newman. Partitions obtained with both techniques are rather consistent, mainly because information centrality has a strong correlation with edge betweenness. The algorithm by Fortunato et al. gives better results when communities are mixed, i.e. with a high degree of interconnectedness, but it tends to isolate leaf vertices and small loosely bound subgraphs.

D. Constructive methods

These are relatively new approaches according to which some heuristics are applied in order to build communities directly without following a hierarchical approach as both agglomerative and separative approaches do. One example is the work presented in K. Steinhäuser et al., 2010, where authors propose an algorithm that weights links in a complex network based on some node similarity metrics allowing then to identify communities by removing links whose weights are under a given threshold. Algorithms proposed in R. R. Khorasgani et al., 2010 and J. D. Shah and T. Zaman, 2010, where communities are built around special nodes called also leaders. However in both algorithms each community is centered around one leader only and a node can follow one community at once. In addition the first algorithm requires the user to provide the number $k$ of communities to identify. Next we will talk about a more representative paper (Rushed kanawati, 2011) using constructive method and which is the modifications of the previous work that has been mentioned above.

The basic idea underlaying the proposed algorithm is that a community is composed of two types of nodes: Leaders and Followers. Roughly speaking, leaders form a subset of nodes (eventually one node) whose removal form the network implies community collapse.
The algorithm functions are as follows. First it searches for nodes in the network that are likely to be leaders in a community. Different node ranking metrics can be used in order to estimate the role of a node. These include the classical centrality metrics. Let L be the set of identified leaders. The list L is then reduced by grouping leaders that are estimated to be in the same community. Let C be the set of identified communities. Each node in the network (a leader or a follower) computes its membership degree to each community in C. A ranked list of communities can then be obtained, for each node, where communities with highest membership degree are ranked first. Next, each node will adjust its community membership preference list by merging this with preference lists of its direct neighbors in the network. Different strategies borrowed from the social choice theory can be applied here to merge the different preference lists. This step is iterated until stabilization of obtained ranked lists at each node. The convergence towards a stable state is function of the applied voting scheme. Lastly, each node will be assigned to top ranked communities in its final obtained membership preference list.

The method we have talked above shows the basic idea of constructive methods. Nearly all of them (K. Steinhaeuser, 2010, D. SHah and T. Zaman, 2010, Rushed Kanawati, 2011) split the nodes in the graph into two kinds of nodes, followers and followees, respectively, even though some works assign different names but in the meaning they are the same.

E. Optimization methods

This consists of applying classical optimization approaches that are guided by some community decomposition quality criteria. Examples are work applying genetic algorithms for community detection (S. Li, Y et al., 2010, D. Jin et al., 2010, C. Shi, Y et al, 2009, My T. Thai, 2011).

Compared with the agglomerative and divisive methods based on hierarchy and the constructive methods, optimization methods are another way to maximize community structure function quality such as modularity. When talking to measuring the community detection algorithm, The former three classes of methods usually do a lot of tests on applications of subgraphs of real-world networks, however, optimization methods focus more on theory analysis and mathematical deduction. From analysis of the properties such as degree distribution in complex networks to prove the ratios of the approximation algorithm and evaluate the performance of the detection algorithm.

In the following part, we will talk about one paper (My T. Thai, 2011) which try to find community structure with performance guarantees in scare-free networks. Scare-free net-
works are complex networks with the property of degree distribution obeying the power law principle.

To mathematically link the power-law topology and the community structure property, they need to select among the wide variety of definitions for communities a quantifiable measurement. The well-known Newmans modularity shall be selected as the only qualification for finding community structure in this paper. Since the introduction of modularity, maximizing modularity has become primal approaches to detect community structure. Despite that existing methods can find sub-optimal solutions in a reasonably fast time, they do not come with any provable solution quality, e.g. the solution is optimal up to a constant factor. Thus, it is desirable to design approximation algorithms to find community structure in the network. Their contribution in this paper is two-fold. They show that the community structure property can be seen as the consequence of the power-law topology in scale-free networks. In addition, we provide an approximation algorithm to find community structure. The algorithm is optimal up to a constant factor when the parameter $\beta > 2$ and a $\log n$ factor when $1 < \beta \leq 2$ in term of maximizing the modularity. It is the first approximation algorithm for the modularity maximization problem. The algorithm encourages further explorations to design both theoretically and empirically justified methods for finding community structure.

First, My T.Thai et al. proposed a following algorithm which belongs to constructive algorithm which is introduced in the previous part. Then a series of analysis is given. They make a detour to focus on the problem of modularity maximization in division of the network into just two communities. The maximum modularity value of the division into two communities are shown to be close to the best possible modularity. Thus, an approximation algorithm for the division into two communities problem also yields an approximation algorithm for the modularity maximization problem.

III. BENCHMARKS TO TEST DETECTION ALGORITHMS

Testing an algorithm essentially means applying it to a specific problem whose solution is known and comparing such solution with that delivered by the algorithm. In the case of graph clustering, a problem with a well defined solution is a graph with a clear community structure. This concept is not trivial, however. Many clustering algorithms are based on similar intuitive notions of what a community is, but different implementations. So it is crucial that the scientific community agrees on a set of reliable benchmark graphs. This mostly applies to computer-generated graphs, where the built-in cluster structure can be arbitrarily designed. In the literature real networks are used as well, in those cases in which communities are well
defined because of information about the system.

We start our survey from computer-generated benchmarks. A special class of graphs has become quite popular in the last years. They are generated with the so-called planted \( l \)-partition model (Condon and Karp, 2001). The model partitions a graph with \( n = g \cdot l \) vertices in \( l \) groups with \( g \) vertices each. Vertices of the same group are linked with a probability \( P_{in} \), whereas vertices of different groups are linked with a probability \( P_{out} \). Each subgraph corresponding to a group is then a random graph with connection probability \( p = P_{in} \). The average degree of a vertex is \( \langle k \rangle = P_{in}(g-1)+P_{out}(l-1) \). If \( P_{in} > P_{out} \) the intra-cluster edge density exceeds the inter-cluster edge density and the graph has a community structure. This idea is quite intuitive and we have encountered it in several occasions in the previous sections. Girvan and Newman considered a special case of the planted \( l \)-partition model (Girvan and Newman, 2002). They set \( l = 4, g = 32 \) (so the number of graph vertices is \( n = 128 \)) and fixed the average total degree \( \langle k \rangle \) to 16. This implies that \( P_{in} + 3 \cdot P_{out} \approx 0.5 \), so the probabilities \( P_{in} \) and \( P_{out} \) are not independent parameters. In calculation it is common to use as parameters \( Z_{in} = P_{in}(g-1) = 31P_{in} \) and \( Z_{out} = P_{out}g(l-1) = 96P_{out} \), indicating the expected internal and external degree of a vertex, respectively. In FIG.5, there is an example of GN benchmark.

![Two states of GN benchmark](From Zhenqing Ye et al., 2008)

Compared with GN benchmark, there is another benchmark which takes self-similarity, which is one property of complex networks, into account. The LFR benchmark is a special case of the planted \( l \)-partition model, in which groups are of different sizes and nodes have different degrees. The node degrees are distributed according to a power law with exponent \( \tau_1 \); the community sizes also obey a power law distribution, with exponent \( \tau_2 \). In the following, \( N \) indicates the number of nodes of the network. In the construction of the benchmark graphs,
each node receives its degree once and for all, and keeps it fixed until the end. In this way, the two parameters pin and pout of the planted l-partition model in this case are not independent. Once the value of pin is set one obtains the value of pout and viceversa. It is more practical to choose as independent parameter the mixing parameter $\mu$, which expresses the ratio between the external degree of a node with respect to its community and the total degree of the node. Of course, in general one may take different values for the mixing parameter for different nodes, but we will assume, for simplicity, that $\mu$ is the same for all nodes, consistently with the standard hypotheses of the planted l-partition model. By construction, the groups are communities when $p_{in} > p_{out}$. This condition can be translated into a condition on the mixing parameter $\mu$. Let us label $k_{in}^i$ and $k_{out}^i$ the internal and external degree of node $i$ with respect to its community (which we denote with $c$). By definition, $k_{in}^i$ is the number of neighbors of $i$ that belong to its community $c$ and $k_{out}^i$ the number of neighbors of $i$ that belong to the other communities. The number of available connections $k_{out}^c (k_{in}^c)$ outside (inside) $c$ is given by the sum of the degrees of the nodes outside (inside) the community. If the numbers of nodes inside and outside $c$ are not too small, the sum of their degrees can be approximated by the product of the average degree $\langle k \rangle$ by the number of nodes. We indicate with $n_c$ the number of nodes of the community $c$ of node $i$, so we have that $k_{out}^c \sim (N - n_c) \langle k \rangle$ and $k_{in}^c \sim (n_c) \langle k \rangle$. By definition of the linking probabilities $p_{in}$ and $p_{out}$ we deduce that

$$P_{out} = \frac{k_{out}^i}{k_{out}^c} = \frac{k_{out}^i}{(N - n_c) \langle k \rangle}$$

and

$$P_{in} = \frac{k_{in}^i}{k_{in}^c} = \frac{k_{in}^i}{n_c \langle k \rangle}$$

In this way, the condition for the existence of communities $p_{in} < p_{out}$ becomes

$$\frac{k_{in}^i}{n_c \langle k \rangle} > \frac{k_{out}^i}{(N - n_c) \langle k \rangle}$$

At last, we can get $\mu$ by

$$\mu < \frac{N - n_c}{N}$$

The condition expressed in the last is general, and applies to any version of the planted l-partition model. When communities are different in size, the upper bound on $\mu$ depends on the specific community at hand. However, if $n_{max} c$ is the size of the largest community, we can safely assume that, whenever $\mu < (N - n_{max} c)/N$, all communities are well defined. In the
GN benchmark, where $nc = 32$ and 128, the condition becomes $\mu < 3/4$. This is interesting, as in most works using the GN benchmark, one usually assumes that communities are there as long as $\mu < 1/2$, whereas they are not well defined for $\mu > 1/2$. Instead, we see that communities are there, at least in principle, up until $\mu = 3/4$. However, we stress that, even if communities are there, methods may be unable to detect them. The reason is that, due to fluctuations in the distribution of links in the graphs, already before the limit imposed by the planted partition model it may be impossible to detect the communities and the model graphs may look similar to random graphs. We notice that, on large networks, when $n_c \ll N$, the limit value of $\mu$ below which communities are defined approaches one. In our tests with the LFR benchmark, we will often be in this regime.

IV. CONCLUSION

When real-world complex networks develops, researchers put more effort on the properties of complex networks rather than the properties of single or a small set of vertices and edges. Community structure is one important property of complex networks and how to construct or detect the good community structure has been and still is a popular topic. However, What the field lacks the most is a theoretical framework that defines precisely what clustering algorithms are supposed to do. Everybody has his/her own idea of what a community is, and most ideas are consistent with each other, but, as long as there is still disagreement, it remains impossible to decide which algorithm does the best job and there will be no control on the creation of new methods. Therefore, we believe that the first and foremost task that the scientific community working on graph clustering has to solve in the future is defining a set of reliable benchmark graphs, against which algorithms should be tested. Up to now, there are a lot of good works including the analysis the of the community structure and therefore there are two class of algorithms which are the agglomerative and divisive, and the constructive algorithms and optimization approach, both of which are quite popular these two years and direct another way to exploit and detect community structure.

REFERENCES


