Scalable and Accurate Graph Clustering and Community Structure Detection

Hristo N. Djidjev
Los Alamos National Labratory
Los Alamos, NM 87545
Email: djidjev@lanl.gov

Melih Onus
Department of Computer Engineering
Bilkent University
Bilkent, Ankara, 06800, Turkey
Email: melih@asu.edu

***Abstract*—One of the most useful measures of cluster quality is the modularity of the partition, which measures the difference between the number of the edges joining vertices from the same cluster and the expected number of such edges in a random graph. In this paper we show that the problem of finding a partition maximizing the modularity of a given graph G can be reduced to a minimum weighted cut problem on a complete graph with the same vertices as G. We then show that the resulting minimum cut problem can be efficiently solved by adapting existing graph partitioning techniques. Our algorithm finds clusterings of a comparable quality and is much faster than the existing clustering algorithms.**

I. Introduction

One way to analyze and understand the information contained in the huge amount of data available on the WWW and the relationships between the individual items is to organize them into "communities," maximal groups of related items. Nowadays networks that need to be analyzed have large sizes with millions of vertices and edges, so traditional analysis with global statistics, for example degree distributions, do not provide information about the structure of the system. Communities are subgraphs containing nodes with similar features, hence detecting communities will disclose such similarities among nodes and also will show how the system is organized [20]. In sociology, community is a tighter and more cohesive social entity. Determining the communities is of great theoretical and practical importance, since they correspond to subgroups of related items such as collaboration networks, groups of friends in online social networks, sets of scientific publications or news stories on a given topic, related commercial items, e.g., movies, books, etc. Communities also arise in other types of networks such as computer and communication networks (the Internet, ad-hoc networks) and biological networks (protein interaction networks, genetic networks). Finding communities in such networks will help in discovering hidden relationships between the nodes, in analyzing the flow of information, and understanding the organization and function of the system.

In distributed and parallel computing, knowing the community structure can be used to partition the data to the processors in a high-performance system. By having all nodes belonging to the same community assigned to the same processor results in data allocations where most of the communication and data exchange occurs within the same processor, thereby increasing the efficiency of the system. Community detection is also relevant to social computing, a very active and fast growing areas of research and technology providing applications and services that facilitate distributed computing by groups such as teams, communities,organizations, and markets [26]. Knowing the communities which are often implicit in the data is key for the efficiency of such systems.

The problem of identifying communities in a network is usually modeled as a *graph clustering* (GC) problem, where vertices correspond to individual items and edges describe relationships. Then the communities correspond to subgraphs with a lot of edges between vertices belonging to the same subgraph (called *in-cluster* edges) and fewer edges between vertices from different subgraphs (called *between-cluster* edges). The GC problem has been intensively studied in the recent years in relation to its applications in the analysis of networks. Girvan and Newman propose in [11], [24] algorithms based on the *betweenness* of the edges of a graph. Betweenness is a characteristic that measures the number of the shortest paths in a graph that use any given edge. In [22] Newman describes an algorithm based on a characteristic of clustering quality called *modularity*, a measure that takes into account the number of in-cluster edges and the expected number of such edges. (We formally define and discuss modularity in more detail in the next section.) Modularity has become a measure of choice for community detection researchers. Its advantages are simplicity and intuitivity of the notion, good accuracy, and computational feasibility (although all known algorithms are approximate). A faster version of the algorithm from [22] was described by Clauset *et al.* in [7]. Several algorithms have been proposed based on other techniques such as computing eigenvectors of the graph Laplacian, e.g., [31], [21], [25], simulated annealing [30], [12], and belief propagation [13]. In all previous cases the algorithms reported in the literature are either not fast enough, or are inaccurate. The problem of finding a partition that maximizes the modularity was shown to be NP-hard [5].

In this paper we will describe a new approach for GC that uses our newly discovered relationship between the GC

A preliminary version of this paper was presented at Fourth Workshop on Algorithms and Models for the Web-Graph (WAW 2006). This work has been supported by the Department of Energy under contract W-705-ENG-36.

and the minimum weighted cut problems. Our algorithm finds clusterings of a comparable quality and is much faster than the existing clustering algorithms. The *general minimum weighted cut* (MWC) problem is, given a graph $G = (V, E)$ with *arbitrary* real weights on its edges, find a partition of V such that the set of all edges of G that join vertices from different sets of the partition, called a *cut* of the partition, is of minimum weight. GC looks related to the MWC problems since, in a good quality clustering, the weight of the edges between different sets of the partition (the cut) should be small compared to the weight of the edges inside the sets. But the MWC problem can not be directly applied to solve the GC problem since it does not take into account the sizes of the subgraphs induced by the cut (e.g., it is likely that the minimum cut will consist of the edges incident to a single vertex). There are some minimum cut based clustering algorithms, e.g., [10], that use maximum flow computations combined with heuristics, but they are typically slower than modularity based algorithms, e.g. [7], and, moreover, they cannot determine the optimal number of clusters and, instead, construct a hierarchical decomposition of the set of all vertices of the graph.

In this paper we prove that the problem of finding a partition of a graph G that maximizes the modularity can be reduced to the problem of finding a MWC of a weighted complete graph on the same set of vertices as G. We then show that the resulting minimum cut problem can be solved by modifying existing fast algorithms for graph partitioning. We demonstrate by experiments that our algorithm has generally a better quality and is much faster than the best existing GC algorithms.

II. Our clustering algorithm

A. Preliminaries

A *graph* G is an ordered pair $(V(G), E(G))$ of sets, where $V(G)$ is the set of the *vertices* and $E(G)$ is the set of the *edges* of G and each edge is an unordered pair (v, w) of vertices. If $E' \subseteq E(G)$, then by $G - E'$ we denote the graph $(V(G), E(G) \setminus E')$. A graph is bipartite, if $E(G) \subseteq \{(v_1, v_2) \mid v_1 \in V_1, v_2 \in V_2\}$, where $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$. A *path* p in G is a sequence $(v_1, \ldots, v_k)$ of vertices such that $(v_i, v_{i+1}) \in E(G)$ for $1 \le i < k$. If $v_1 = v_k$, then p is a *cycle*. G is *connected* if there is a path between any pair of vertices of G. The *components* of G are its maximal connected subgraphs. A *partition* $\mathcal{P}$ of G is a division of $V(G)$ into subsets $V_1, \ldots, V_s$ such that $V_i \cap V_j = \emptyset$ for $i \neq j$ and $\bigcup_{i=1}^{s} V_i = V(G)$. If $s = 2$, then $\mathcal{P}$ is a *bisection*. Note that, in contrast to other works, our definitions of partition and bisection does not require the parts to be balanced in size. A set $C \subseteq E(G)$ is a *cut* of G if there exists a partition $\mathcal{P}$ of G such that C is the set of the edges of G joining vertices from different sets of $\mathcal{P}$. We will use the notation $C = \mathrm{cut}(\mathcal{P})$ and $\mathcal{P} = \mathrm{part}(C)$ and define $\mathrm{cut}(V_i, V_j) = \{(t, u) \in E(G) \mid t \in V_i, u \in V_j\}$. Two partitions $\mathcal{P}_1$ and $\mathcal{P}_2$ are *equivalent*, if $\mathrm{cut}(\mathcal{P}_1) = \mathrm{cut}(\mathcal{P}_2)$. If there are weights $\mathrm{wt}(\cdot)$ associated with the edges of G, then by $\mathrm{cutWt}(\mathcal{P}) = \mathrm{wt}(C)$ we denote the sums of the weights of all edges in C. If M is a finite set, by $|M|$ we denote the number of the elements of M.

B. Modularity optimization as a minimum cut problem

As there is no formal definition of clustering and what the clusters of a given graph are, in general it is not possible to determine if a certain partition represents the ”correct” clustering or which of two alternative partitions of a graph corresponds to a better clustering. For that reason, researchers have used their intuition to define measures for cluster quality that can be used for comparing different partitions of the same graph. One such measure, introduced in [23], [24], which has received considerable attention recently, is the *modularity* of a graph. Given an n-vertex m-edge graph $G = (V(G), E(G))$ and a partition $\mathcal{P}$ of $V(G)$ into k subsets (clusters) $V_1, \ldots, V_k$, a random graph distribution $\mathcal{G}$ on $V(G)$, the modularity $Q(\mathcal{P}, G, \mathcal{G})$ of $\mathcal{P}$ with respect to $\mathcal{G}$ (or $Q(\mathcal{P})$ for short if G and $\mathcal{G}$ are clear from the context) is a number between -1 and 1 defined as

$$Q(\mathcal{P}) = Q(\mathcal{P}, G, \mathcal{G}) = \frac{1}{m} \sum_{i=1}^{k} (|E(V_i)| - \mathrm{Ex}(V_i, \mathcal{G})),$$

where $E(V_i)$ is the set of all edges of G with endpoints in V_i and $\mathrm{Ex}(V_i, \mathcal{G})$ is the expected number of such edges in a random graph with a vertex set V_i from a given random graph distribution $\mathcal{G}$ on $V(G)$. $Q(\mathcal{P})$ measures the difference between the number of in-cluster edges and the expected value of that number for $\mathcal{P}$ in a random (e.g., without cluster structure) graph on the same vertex set. Larger values of $Q(\mathcal{P})$ correspond to better clusterings.

Having the definition of $Q(\mathcal{P})$, we can formulate the clustering problem as finding a partition $\mathcal{P} = \{V_1 \cup \cdots \cup V_k\}$ of $V(G)$ such that

$$\sum_{i=1}^{k} (\, |E(V_i)| - \mathrm{Ex}(V_i, \mathcal{G})) \to \max . \tag{1}$$

Clearly

$$\max_{\mathcal{P}} \{ \sum_{i=1}^{k} (\, |E(V_i)| - \mathrm{Ex}(V_i, \mathcal{G}) \,) \}$$

$$= - \min_{\mathcal{P}} \{ \, - \sum_{i=1}^{k} (\, |E(V_i)| - \mathrm{Ex}(V_i, \mathcal{G}) \,) \}$$

$$= - \min_{\mathcal{P}} \{ \, (|E(G)| - \sum_{i=1}^{k} \, |E(V_i)| \,) - (|E(G)| - \sum_{i=1}^{k} \, \mathrm{Ex}(V_i, \mathcal{G}) \,) \}.$$

Denote

$$\mathrm{ExCut}(\mathcal{P}, G, \mathcal{G}) \} = |E(G)| - \sum_{i=1}^{k} \, \mathrm{Ex}(V_i, \mathcal{G}) \, .$$

Intuitively, $\mathrm{ExCut}(\mathcal{P}, G, \mathcal{G})$ is the expected value of $|\mathrm{cut}(\mathcal{P})|$ with respect to the random graph class $\mathcal{G}$, assuming the expected number of edges for $\mathcal{G}$ is $|E(G)|$. Then

$$\max_{\mathcal{P}} \{ \, \sum_{i=1}^{k} (\, |E(V_i)| - \mathrm{Ex}(V_i, \mathcal{G}) \,) \} =$$

$$-\min_{\mathcal{P}}\{\ |\text{cut}(\mathcal{P})| - \text{ExCut}(\mathcal{P}, G, \mathcal{G})\}.$$

Hence, instead of problem (1), one can address the problem of computing

$$\operatorname*{argmin}_{\mathcal{P}}\{\ |\text{cut}(\mathcal{P})| - \text{ExCut}(\mathcal{P}, G, \mathcal{G})\}\,. \tag{2}$$

The last expression shows that we can solve (1) as a problem of finding a MWC in a complete graph G' with a vertex set $V(G)$ and weight $\text{weight}(i,j)$ on any edge $(i,j) \in E(G')$ defined by

$$\text{weight}(i,j) = \begin{cases} 1 - p_{ij}, & \text{if } (i,j) \in E(G) \\ -p_{ij}, & \text{if } (i,j) \notin E(G), \end{cases} \tag{3}$$

where p_{ij} is the probability that there is an edge between vertices i and j in a random graph from the class $\mathcal{G}$. Then, problem (1) is equivalent to the problem of computing

$$\operatorname*{argmin}_{\mathcal{P}'}\{\text{cutWt}(\mathcal{P}')\}\,, \tag{4}$$

where $\text{cutWt}(\mathcal{P}')$ denotes the weight of the cut of $\mathcal{P}'$.

We summarize these observations in the following theorem.

Theorem 1. *The problem of finding a partition of a graph $G = (V, E)$ that minimizes the modularity can be reduced in $O(\,|V| + |E|\,)$ time to the problem of finding a minimum weighted cut in a complete graph $G' = (V, E')$ with edge weights given by (3).*

For the reduction time bound in Theorem 1 we assume that the edges of $E' \setminus E$ are defined implicitly. There are several choices for $\mathcal{G}$ that have been favored by various researchers. The random graph model $G(n,p)$ of Erdös-Renyi [8] defines n vertices and puts an edge between each pair with probability p. Clearly, the expected number of edges of $G(n,p)$ is $\binom{n}{2}p$. Hence, for a graph with expected number of edges m

$$p_{ij} = p = \frac{m}{\binom{n}{2}}\,. \tag{5}$$

One disadvantage of the $G(n,p)$ model is that it fails to capture important features of the real-world networks, in particular, the degree distribution. As has been recently observed [3], many important types of networks like technological networks (the Internet, the WWW), social networks (collaboration networks, online social networks), biological networks (protein interactions) have degree distributions that follow a *power law*, e.g., the fraction of the vertices that have degree $k > 0$ is roughly proportional to $\alpha k^{-\lambda}$ for some constants α and $\lambda > 0$. Such networks are called *scale-free*. In comparison, the degrees of a random graph from the $G(n,p)$ model follow a Poisson distribution, i.e., the probability that a given vertex has degree k is $\binom{n}{k}p^k(1-p)^{n-k}$ and the expected degree of each vertex is pn. Hence, the Erdös-Renyi model may not be suitable as a choice for $\mathcal{G}$ when used for determining the community structure of graphs of the above type.

One model that takes into account the degrees of the vertices is studied by Chung and Lu in [6]. In that model, the probability that there is an edge between a vertex i and a vertex j is

$$p_{ij} = \frac{d_i d_j}{\sum_{k=1}^{n} d_k}\,, \tag{6}$$

where $d_1, \cdots, d_n$ are positive reals corresponding to the degrees of the vertices such that $\max_{1\le i\le n} d_i^2 < \sum_{i=1}^{n} d_i$. (The last condition guarantees that such a graph exists if all numbers d_i are integers and will be always satisfied if numbers d_i are chosen to be the degrees of G.) We will refer to that model as the Chung-Lu (CL) model. Clearly, in the CL model, the expected degree of vertex i is d_i, compared with pn (i.e., independent on i) in the $G(n,p)$ model. CL model captures the degree distribution of the real world networks, since the expected degree of vertex i is d_i.

Note that for both of the above choices of $\mathcal{G}$ the expected number of edges for a graph in $\mathcal{G}$ is $|E(G)|$.

In the next section we will describe an efficient method for finding a MWC of a complete graph G' with weights on the edges satisfying (3) and p_{ij} defined by (5) or (6).

C. Finding a MWC using multilevel graph partitioning

Above we established an important relationship between the modularity optimization and the MWC problems, i.e., that the problem of finding a partition of a given graph that maximizes the modularity can be reduced to the problem of finding a minimum weight cut. Most existing work on the MWC problem considers the case where all weights are non-negative. The MWC problem in the case of non-negative weights is known to be polynomially solvable, e.g., by using algorithms for computing maximum flows [1]. In contrast, the MWC problem in case of real-value weights is NP-hard and algorithmic aspects of the problem are much less studied. Here we show that available heuristics for another related problem, graph partitioning, can be adapted to solve this version of the MWC problem.

1) Overview of the multilevel graph partitioning.: Formally, the *graph partitioning* (GP) problem is, given a graph $G = (V, E)$, to find a partition (V_1, V_2) of V such that $||V_1| - |V_2|| \le 1$ (i.e., the partition is *balanced*) and $\text{cut}(V_1, V_2)$ is minimum. (Some versions of the problem consider partitions into an arbitrary number of parts.) Hence, in comparison with the minimum cut problem, there is the additional requirement for a balanced partition. Because of its important applications, e.g., in high performance computing and VLSI design, GP is a well-researched problem for which very efficient methods have been developed. Spectral partitioning methods [27], [28] produce good results but have long running times. Geometric graph partitioning algorithms [14], [29] can be applied only if one has the coordinates for the vertices. Multilevel graph partitioning schemes [4], [15] can produce better partitions than spectral methods and have better running times for most of the instances. The multilevel GP is both fast and accurate for a wide class of graphs that appear in practical applications. It is inspired by the multigrid method from computational mathematics. The multigrid method is a group of algorithms

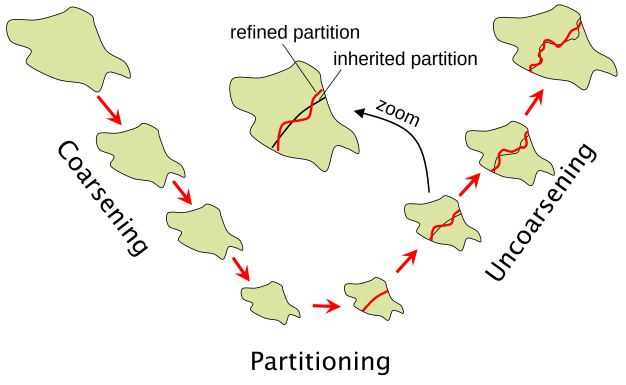

Fig. 1. The stages of multilevel partitioning.

for solving differential equations using a hierarchy of discretizations. The multilevel GP has been used in the works of Barnard and Simon [4], Hendrickson and Leland [15], Karypis and Kumar [17], [18], and others. The method for bisecting a graph consists of the three phases(Figure 1).

Thus, we get a sequence of graphs $G = G_0, G_1, G_2, \dots, G_k$. We partition G_k by greedy graph growing partitioning. At the refinement phase, we project partitioning P_i of G_i to P_{i-1} of G_{i-1}. We have more degrees of freedom at G_i than G_{i-1} and we improve P_i using Kerninghan-Lin algorithm.

Coarsening phase. The original graph G is coarsened by partitioning it into connected subgraphs and contracting them to single vertices. For that purpose, a maximal matching M of the vertices of G is constructed. Such a matching is computed in a greedy manner, i.e., for each unvisited unmatched vertex v one of its unmatched neighbors is picked at random; if none, then v remains unmatched. Then each edge of M is collapsed to a single vertex, replacing any set of resulting parallel edges by a single edge. Moreover, a weight of each new vertex (respectively edge) is assigned equal to the sum of the weights of the vertices (respectively edges) that it represents. Weights on the original vertices of G are defined 1, in the case of the $G(n,p)$ model, or their degrees, in the case of the CL model, as justified in Corollary 1 below. (The coarsening procedure, including alternative methods for determining the set of the shrunk subgraphs and analysis of their effect on the quality of the final partition, is described in much detail in [17].) The resulting graph is coarsened repeatedly by the same procedure until one gets a graph of a sufficiently small size. Let $G_0 = G, G_1, \dots, G_l$ be the resulting graph sequence.

Partitioning phase. The graph G_l is partitioned into two parts using any available partitioning method (e.g., spectral partitioning or the Kernighan-Lin (KL) algorithm [19]).

Uncoarsening and refinement phase. The partition of G_l is projected on G_{l-1}, which involves replacing each vertex by the subgraph that was contracted to that vertex during the coarsening step. Since the weight of each vertex of G_l is a sum of the weights of the corresponding vertices of G_{l-1}, then the cut of both partitions will have the same weight. However, since G_{l-1} has more vertices than G_l, it has more degrees of freedom and, therefore, it is possible to refine the partition of G_{l-1} in order to reduce its cut size. To this end, the projection of the partition of G_l is followed by a refinement phase, which is usually based on the KL algorithm (see next subsection). In the same way, the resulted partition of G_{l-1} is converted into a partition of G_{l-2} and refined, and so on until a partition of G_0 is found.

2) Kernighan-Lin refinement.: Since the refinement step is the most involved part of the algorithm, which ultimately determines its accuracy and efficiency, we will describe it in more detail. It has been shown [17] that the KL algorithm can be a good choice for performing the refinement.

The KL algorithm involves several iterations, each consisting of moving a vertex from one set of the partition to the other. Let $\mathcal{P} = \{P_1, P_2\}$ be the current partition. For each vertex u of the graph a *gain* for u is defined as

$$gain(u) = \sum_{v \in N(u) \setminus P(u)} weight(u,v) - \sum_{v \in N(u) \cap P(u)} weight(u,v), \tag{7}$$

where $N(u)$ is the set of all neighbors of u and $P(u)$ is that set of $\mathcal{P}$ that contains u. $gain(u)$ measures how the weight of the cut will be affected if u is moved from $P(u)$ to the other set of $\mathcal{P}$. The KL algorithm then selects a vertex w from the smaller set of the partition with a maximum gain, moves it to the other set, and updates the gains of the vertices adjacent to w. Moreover, w is marked so that it will not be moved again during that refinement step. The process is continued until either all vertices have been moved, or the S most recent moves have not led to a better partition. (S is a user chosen parameter that is set to 50 in the current implementation.) At the end of the refinement step, the last $s \leq S$ moves that have not improved the partition are reversed.

3) Implementation: The implementation of our algorithm for clustering is based on the version of multilevel partitioning implemented by Karypis and Kumar [17], [18], which has been made freely available as a software package under the name METIS. The multilevel graph partitioning algorithm is also implemented in parallel by Karypis and Kumar [16] and is called ParMetis. ParMetis is an MPI-based parallel library. The algorithms implemented in ParMetis are based on the parallel multilevel k-way graph partitioning, adaptive repartitioning, and parallel multi-constrained partitioning schemes. Initially, a single processor partitions the original graph into two subgraphs, then two processors partition the two subgraphs just produced, and so on. In our experiments, we used METIS.

Note that graph partitioning, minimum cut, and clustering are related, but with important differences, problems, as illustrated in Table I. We have already shown how the clustering problem can be reduced to a minimum cut problem and here we will show how the resulting minimum cut problem can be solved by a graph partitioning algorithm based on METIS. Because of the differences between graph partitioning and MWC, we have to make some evident changes. For instance, since graph partitioning requires balanced partitions, we have to drop the requirement for balance of the partition. At the uncoarsening phase, we ignore the restrictions that control the sizes of the parts. We have also to determine the cardinality of the partition that minimizes the cut size. But the main implementation difficulty is related to the size of G'. Although the original graph, G, is often sparse, i.e., it has n vertices and $O(n)$ edges, the transformed one, G', is always dense, as it has $\binom{n}{2} = \Omega(n^2)$ edges. The main challenge will be to construct an algorithm whose complexity is close to linear on the size of the original graph, rather than on the size of the transformed one. Next we show that it is possible to simulate an execution of a KL refinement step on G' by explicitly maintaining information only about the edges from the original graph G and implicitly taking into account the remaining edges by modifying the formulae for computing weights and gains.

In order to give intuition about why this works, assume that the edges of G' belong to two types that we call *visible* and *invisible*. The visible edges correspond to the edges of the original graph G and are therefore few (assuming G is sparse). These edges carry weight 1 and are maintained explicitly. The invisible edges are between any two vertices of G'. (Note that for each visible edge there is also an invisible one parallel to it, i.e., joining the same endpoints.) The weight of invisible edge (i, j) is $-p_{ij}$. Although the number of invisible edges is $\Omega(n^2)$, because of their uniform distribution, the contribution of these edges to the cut is easy to compute by maintaining additional information of size $O(1)$ only. The next two lemmas formalize this notion.

Lemma 1. *Let $\mathcal{P} = \{V_1, V_2\}$ be a partition of G and let G' be the transformed weighted graph with respect to the $G(n,p)$ random graph model. Let $\mathcal{P}'$ be the cut in G' corresponding to $\mathcal{P}$. Then $\mathrm{cutWt}(\mathcal{P}') = \mathrm{cutWt}(\mathcal{P}) - |V_1|\,|V_2|p$, where $p = m/\binom{n}{2}$.*

Proof: Follows from formulae (3) and (5). There is an edge in G' joining any vertex from V_1 with any vertex in V_2. For an edge from G the corresponding weight is $1 - p$, and an edge in G' not in G the corresponding weight is $-p$. ■

The lemma shows that if one maintains the values of $|V_1|$ and $|V_2|$ during a KL refinement, one can work with the original graph G rather than with the modified G', updating at each step the value of the cut in $O(1)$ time using Lemma 1.

A similar formula holds for the case of the CL model.

Lemma 2. *Let $\mathcal{P} = \{V_1, V_2\}$ be a partition of G and let G' be the corresponding weighted graph with respect to the CL random graph model. Assign a weight $\mathrm{wt}(v)$ to each vertex v equal to its degree. Let $\mathcal{P}'$ be the cut in G' corresponding to $\mathcal{P}$. Then*

$$\mathrm{cutWt}(\mathcal{P}') = \mathrm{cutWt}(\mathcal{P}) - \mathrm{wt}(V_1)\mathrm{wt}(V_2)p\,, \qquad (8)$$

where $\mathrm{wt}(V_i) = \sum_{v\in V_i} \mathrm{wt}(v)$ and $p = \left(\sum_{v\in V(G)} \mathrm{wt}(v)\right)^{-1}$.

Proof: Follows from formulae (3) and (6) and the equality

$$\sum_{v\in V_1}\sum_{w\in V_2} \frac{\mathrm{wt}(v)\mathrm{wt}(w)}{p} = \left(\sum_{v\in V_1} \mathrm{wt}(v)\right)\left(\sum_{w\in V_2} \mathrm{wt}(w)\right)/p\,.$$

■

According to the lemma, the cut weight of $\mathcal{P}'$ can be computed in $O(1)$ time given the cut weight of $\mathcal{P}$, if one maintains the values of the weights of V_1 and V_2 during the KL refinement.

In the case of both the $G(n,p)$ and the CL random graph models, for moving a vertex v from one partition to another during a KL refinement we need only to update the gains of the neighbors of v in G. Having those gains, one can maintain $\mathrm{cutWt}(\mathcal{P})$ in total time proportional to the size of G, excluding the time for priority queue operations needed to extract vertices with maximum gains, which is $O(n \log n)$ in total. By Lemma 1 or Lemma 2, one can at any time compute $\mathrm{cutWt}(\mathcal{P}')$ from $\mathrm{cutWt}(\mathcal{P})$ and the weights of the partitions in $O(1)$ additional time.

From Lemma 1 and Lemma 2 it follows that in the case of both models the same KL refinement algorithm can be used, if the vertex weights are appropriately defined.

Corollary 1. *Let $\mathcal{P} = \{V_1, V_2\}$ be a partition of G and let G' be the corresponding weighted graph with respect to either the $G(n,p)$ or the CL random graph model. Define the weight of any vertex v to be 1, in the case of the $G(n,p)$ model, or the degree of v, in the case of the CL model. Then $\mathrm{cutWt}(\mathcal{P}')$ can be computed by formula (8).*

4) Clustering into an optimal number of clusters: The algorithm described above is a bisection algorithm, i.e., it finds a partition (and hence clustering) of the input graph into two parts. Our algorithm for an arbitrary number of clusters uses the following recursive procedure. We run the bisection algorithm described above and let P be the resulting partition. If P consists of only one set (i.e., the original graph G does not have a good cluster partition), we are done. Else, we run recursively the bisection algorithm on the two subgraphs G_1 and G_2 of G induced by the vertices of the two sets of P. It is important to keep, during that recursive call, the weights of the edges computed during the first iteration instead of recomputing them based on G_1 and G_2. The reason is that the random graph model based on G will be different than those based on G_1 and G_2 since formulae 5 and 6 will produce different values for p_{ij}.

5) Time analysis.: The Fiduccia and Mattheyses of the KL algorithm from [9] takes linear time. It uses an array of bucket lists to achieve linear running time. By using the analysis of Fiduccia and Mattheyses of the KL algorithm from [9], it follows that clustering any network of n vertices and m edges into two communities by our algorithm takes $O(n \log n + m)$ time, where n and m are the numbers of the nodes and links of the network, respectively. Finding a clustering in optimal number of k parts takes $O((n \log n+m)d)$ time, where d is the depth of the dendrogram describing the clustering hierarchy. Since the dendrogram is represented by a binary tree, $\log_2 k \le d \le k$.

Problem	Modularity optimization	Minimum Cut	Graph Partitioning
Objective	Minimize modularity	Minimize cut size	Minimize cut size
Balance of partition	Sizes may differ	Sizes may differ	Equal sizes
Cardinality of partition	To be computed	To be computed	An input parameter

TABLE I
COMPARISON BETWEEN MODULARITY OPTIMIZATION, MINIMUM CUT, AND GRAPH PARTITIONING PROBLEMS.

III. EXPERIMENTS AND PERFORMANCE EVALUATION

We performed a number of experiments on real graphs as well as on randomly generated graphs, in order to measure the accuracy of our algorithm and its efficiency as well as to compare it with previous algorithms. We first describe several experiments on real-world graphs, whose purpose is to illustrate the use of our algorithm and community detection in general for extracting structural information from network data. Then we include the results of an experiment measuring the algorithm accuracy, the so called Newman-Girvan test. We include this test as an example of non-modularity based accuracy test and because of its popularity. Its disadvantages are that it uses graphs of very special structure and of relatively small sizes. That is why we concentrate most of our effort and describe in most detail the results of another type of experiments, included in the third subsection. It uses graphs of different size and structures, and on which we are able to test both the speed and the accuracy of our algorithm versus several others. In all experiments the CL version of our algorithm was used.

A. Testing on real-world-data graphs

We tested our algorithms on a number of real-world graphs such as the *nd.edu* domain data [2], the United States college football data [11], and the Zachary's karate club network [32]. In all cases our algorithms produced clustering consistent with our previous knowledge about the communities. We describe here in more detail the Zachary club network only. The goal of including this example is not to demonstrate the qualities of our algorithm, but rather to illustrate the notion of community detection with a real-life example. This network, used often for illustration of community detection algorithms, describes the interactions between the members of a karate club that consequently split into two because of infight between the members, thereby revealing the hidden communities of the original network. The vertices of the corresponding graph denote club members and the edges correspond to friendships between members. As shown on Figure 2, the bisection constructed by our algorithm classified correctly the members of the two subgroups, except for node 10. That node has the same number of links (one) to both communities, hence adding it to the smaller community results in a greater modularity (i.e., our bisection has a better modularity than the "real" one.)

B. Newman-Girvan accuracy test

Following the experimental setting of [24], we generated random graphs with 128 vertices and 4 communities of size

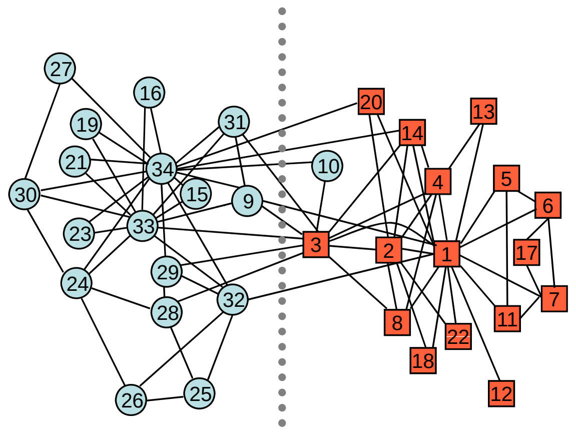

Fig. 2. Zachary's karate club network. Members of the communities resulting after the split are denoted by circles and squares, respectively. The communities found by our algorithm are separated by the vertical line.

Outdegree	Degree	Newman-Girvan	Ours
1	16	1.00	1.00
2	16	1.00	1.00
3	16	0.98	1.00
4	16	0.97	1.00
5	16	0.95	1.00
6	16	0.85	0.99
7	16	0.60	0.95
8	16	0.30	0.79

TABLE II
COMPARING THE QUALITY OF THE CLUSTERING OF OUR ALGORITHM AND NEWMAN-GIRVAN'S ALGORITHM.

32 each. The expected degree of any vertex is 16, but the expected *outdegree* (the expected number of neighbors of a vertex that belong to a different community) is set to i in the i-th experiment ($i \leq 16$). Hence, higher values of i correspond to graphs with weaker cluster structures. The experiment is intended to measure the sensitivity of the algorithm to the strength of the communities.

In order to decide whether to include an edge (v, w) in the graph in the i-the experiment, a random number r in the interval $[0, 1]$ is generated and (v, w) is accepted if $r \geq i/31$ and v and w belong to the same community or $r \geq (16-i)/96$ and v and w belong to different communities, and is rejected otherwise.

Table II compares the quality of the clusterings produced by Newman-Girvan's algorithm and ours. A clustering produced by any of the algorithms is considered "correct" if it matches the original partition of communities from the graph generation phase. (Note that, due to the probabilistic nature of the graphs, the clustering that maximizes the modularity might be different from the original partition, especially if the modularity is low.)

Our algorithm classifies correctly more than 99% of the edges for outdegrees $0, 1, 2, 3, 4, 5, 6$ and in all cases it is better than Newman-Girvan's (more than twice better for the case $\text{outdegree} = 8$).

C. Testing speed and accuracy

Table I (supplementary file) compares the performance of our algorithm with four other algorithms that are considered among the best with respect to their speeds and/or accuracies. Clauset, Newman, and Moore's algorithm [7] is an *agglomerative* algorithm that is an improvement of a previous algorithm [22] in terms of the speed and is claimed to have the same quality of the partition. Agglomerative algorithms start with a community partition, where each single vertex represents a community. At each iteration a pair of communities are merged into a single one such that a measure of cluster quality, in this case the modularity, is improved. The second algorithm is Newman's algorithm described in [25], which is a spectral algorithm based on eigenvector computations. The other two algorithms, of Guimera and Amaral [12] and Reichardt and Bornholdt [30], are based on simulated annealing optimization.

Most of the algorithms tested, notably Guimera-Amaral and Reichardt-Bornholdt algorithms, have parameters that can be played with in order to improve the accuracy of the algorithms on particular graphs. It is possible that by varying the parameters from experiment to experiment and from graph to graph, the quality of some partitions would have improved. Our algorithm also has parameters that allow trading off speed for accuracy. However, such type of optimization and fine-tuning of the algorithms is beyond the scope of this paper. In all experiments, we have used the recommended or default values of all parameters.

The test graphs in our experiments are random graphs with varying numbers of clusters, sizes, densities, cluster sizes, and modularities. The graphs are generated by initially assigning a set of isolated vertices into a number of clusters with preset sizes. Then, for each pair of vertices v and w, an edge (v, w) is generated with probability p_{in}, if v and w belong to the same cluster, and with probability p_{out}, otherwise, where p_{in} and p_{out} are input parameters. Increasing p_{in} and/or decreasing p_{out} produces test graphs with better community structures (higher modularities). Increasing both p_{in} and p_{out} increases the density (average vertex degree) of the graph. Hence, varying p_{in} and p_{out} we can create graphs with desired properties.

Experiment 1–10 have been run 100 times on different random graphs and experiments 11–13 have been run 10 times. All experiments have been run on an Intel Xeon CPU 1.60GHz processor desktop computer with 4G of memory.

For each experiment, the table shows the number of the vertices and the average number of edges of the test graph, the number of the clusters in the original partition during generation, and the average modularity of that partition. Then, for each of the algorithms, the average running time and modularity of the partition are listed.

Figures 1 and 2 (supplementary file) show the distribution of the degrees of the vertices for each of the experiments. The *in-degree* of a vertex in those tables is defined here as the number of the adjacent vertices from the same cluster as defined during the generation process and the *out-degree* as the number of adjacent vertices from a different partition. One would expect that the support interval (the interval where the density function is positive) for the in-degrees will always be greater than (to the right of) the one for the out-degrees, in order to have well defined community structures, but this is not always the case. When the number of the communities is large (as in experiment 4), it is possible for the out-degrees of vertices to exceed their in-degrees, while the average number of neighbors to any *fixed* neighboring community to be still lower than the in-degree. In experiments 8, 9, and 10, this effect is further amplified by the low modularity of the partitions, which translates into weaker community structures.

Experiments 1–4 study how the performance of the algorithms depends on the number of clusters, which vary from 2 to 9. The results indicate that the qualities of the clusterings are comparable, while Newman's (N) and Guimera-Amaral's (GA) algorithms time performance is more sensitive to the number of the clusters.

In experiments 5–7, the test graphs have the same numbers of vertices, numbers of cluster, and modularities, but different densities. All algorithms were quite accurate and showed little variance in their performance when sparsity changes.

In experiments 8–10, we compare the algorithms when the modularity (the quality of the original clustering) is low. In these experiments, the Clauset, Newman, and Moore's (CNM) algorithm considerably underperformed the other four with respect to the quality of the partition.

Finally, in experiments 11-13, we compared the scalability of the algorithms. Because of the low scalability of some algorithms and the long time it takes to run a single experiment, those experiments were run only 10 times. As such, small differences in the modularity should be taken with caution, and attention should be paid to the running times, which vary significantly from algorithm to algorithm. The experiments show that the GA algorithm is the slowest, followed by the other simulated annealing based Reichardt and Bornholdt's (RB) algorithm, which is about 4 times faster. Neither of these two algorithms can be used in reasonable time for graphs containing more than a few hundred thousand edges. Algorithm N is much faster than those two and can be used for graphs of size several million edges. The only algorithm that can scale to graphs of sizes up to tens of millions of edges is the CNM algorithm, but it is also the least accurate of all, as seen in the sensitivity tests (experiments 8–10). Yet, our algorithm is about 30 times faster than the CNM algorithm.

Since Table I (supplementary file) shows only averages, we give on Figures 3 and 4 (supplementary file) the distribution of the modularities for each algorithm and each experiment, represented as differences between the modularities of our algorithm and those of the other algorithms. Those figures show that in experiments 1 through 8 our algorithm not only produces equal or better quality clusterings on average, but virtually on any single graph in those tests. The only experiments where the quality is worse in some instances, in spite of the good quality of our algorithm on average, are experiments 9 and 10, where the modularity is very low – 0.123 and 0.081, respectively. Algorithm N performs the best in those experiments, which shows that it can be a good alternative to our algorithm in low to moderate size graphs (up to 4-5 million edges).

In summary, in those experiments our algorithm produced partitions of quality comparable to the most accurate existing algorithms, in times orders of magnitude smaller. Ours is the only one of the tested algorithms that can produce high quality clusterings on graph of sizes exceeding several million edges.

IV. Conclusion

This paper proposes a new approach for modularity optimization by reducing it to a minimum cut problem and then solving the latter problem by applying methods for graph partitioning. Our proof-of-concept implementation, based on the METIS partitioning package, demonstrated the practicality of the approach. The changes we made to METIS were relatively small and various improvements and refinements that take into account the specifics of the clustering problem, use alternative minimum cut or graph partitioning algorithms, or apply heuristics and parameter adjustments in order to improve the accuracy are possible and will be topics of further research.

Acknowledgement. The authors would like to thank the developers of METIS for making their source code publicly available and to Mark Newman, Jörg Reichardt, and Roger Guimera for providing the codes of their algorithms and for helpful comments.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows : theory, algorithms, and applications*. Prentice Hall, 1993.

[2] R. Albert, H. Jeong, and A. L. Barabási, "Diameter of the world wide web," *Nature*, vol. 401, pp. 130–131, September 1999.

[3] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, p. 509, 1999. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/9910332

[4] S. T. Barnard and H. D. Simon, "A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," *Concurrency: Practice and Experience*, vol. 6, pp. 101–107, 1994. [Online]. Available: citeseer.ist.psu.edu/barnard94fast.html

[5] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner, "On modularity clustering," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 2, pp. 172–188, 2008.

[6] F. Chung and L. Lu, "Connected components in random graphs with given degree sequences," *Annals of Combinatorics*, vol. 6, pp. 125–145, 2002.

[7] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Physical Review E*, vol. 70, p. 066111, 2004. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0408187

[8] P. Erdos and A. Renyi, "On random graphs," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.

[9] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *IEEE Design Automation Conference*, 1982, pp. 175–181.

[10] G. Flake, R. Tarjan, and K. Tsioutsiouliklis, "Graph clustering and minimum cut trees," *Internet Mathematics*, vol. 1, pp. 385–408, 2004.

[11] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proc. Natl. Acad. Sci. USA*, vol. 99, pp. 7821–7826, 2002. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0112110

[12] R. Guimera and L. A. N. Amaral, "Functional cartography of complex metabolic networks," *Nature*, vol. 433, p. 895, 2005. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:q-bio/0502035

[13] M. B. Hastings, "Community detection as an inference problem," *Phys.Rev.E*, vol. 74, p. 035102, 2006. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0604429

[14] M. T. Heath and P. Raghavan, "A cartesian parallel nested dissection algorithm," *SIAM J. Matrix Anal. Appl.*, vol. 16, pp. 235–253, 1995.

[15] B. Hendrickson and R. W. Leland, "A multi-level algorithm for partitioning graphs," in *ACM/IEEE Conference on Supercomputing*, 1995. [Online]. Available: citeseer.ist.psu.edu/hendrickson93multilevel.html

[16] G. Karypis and A. Kumar, "A parallel algorithm for multilevel graph partitioning and sparse matrix ordering," in *International Parallel Processing Symposium*, 1996, pp. 314–319.

[17] G. Karypis and B. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.

[18] G. Karypis and V. Kumar, "Multilevel graph partitioning schemes." in *International Conference on Parallel Processing*, 1995, pp. 113–122.

[19] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Sys. Tech. J.*, vol. 49, no. 2, pp. 291–308, 1970.

[20] A. Lancichinetti and S. Fortunato, "Limits of modularity maximization in community detection," *arXiv 1107.1155v1*, 2011.

[21] E. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Physical Review E*, vol. 74, p. 036104, 2006. [Online]. Available: doi:10.1103/PhysRevE.74.036104

[22] J. Newman, "Fast algorithm for detecting community structure in networks," *Physical Review E*, vol. 69, p. 066133, 2004. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0309508

[23] M. Newman, "Mixing patterns in networks," *Physical Review E*, vol. 67, p. 026126, 2003. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0209450

[24] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, p. 026113, 2004. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0308217

[25] M. Newman, "Modularity and community structure in networks," *Proc. Natl. Acad. Sci. USA*, vol. 103, p. 8577, 2006. [Online]. Available: doi:10.1073/pnas.0601602103

[26] M. Parameswaran, "Social computing: An overview," *Communications of the Association for Information Systems*, vol. 19, pp. 762–780, 2007.

[27] A. Pothen, H. D. Simon, and K. P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *SIAM J. Matrix Anal. Appl.*, vol. 11, pp. 430–452, 1990.

[28] A. Pothen, H. D. Simon, L. Wang, and S. T. Bernard, "Towards a fast implementation of spectral nested dissection," *Supercomputing*, pp. 42–51, 1992.

[29] P. Raghavan, "Line and plane separators," *Tech. report. uiucdcs-r-93-1794, Dept. of CS, University of Illinois, Urbana*, 1993.

[30] J. Reichardt and S. Bornholdt, "Detecting fuzzy community structures in complex networks with a potts model," *Physical Review Letters*, vol. 93, p. 218701, 2004. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0402349

[31] S. White and P. Smyth, "A spectral clustering approach to finding communities in graph," in *Proceedings of the SIAM International Conference on Data Mining*, 2005. [Online]. Available: citeseer.ist.psu.edu/734075.html

[32] W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of Anthropological Research*, vol. 33, pp. 452–473, 1977.

Hristo N. Djidjev received the M.S. degree in mathematics and the Ph.D. degree in computer science from Sofia University, Bulgaria. He worked as an assistant professor at Rice University between 1992 and 1998. He is currently a technical staff member at Los Alamos National Laboratory, NM. His research interests are in the areas of combinatorial algorithms, network analysis, sensor networks and algorithm engineering.

Melih Onus received the B.S. degree in computer engineering from Bilkent University, Ankara,Turkey, in 2003, and the Ph.D. degree in computer science from Arizona State University (ASU), Tempe, in 2009. He worked as an instructor at TOBB University of Economics and Technology between September 2009 and April 2011. He is currently an Instructor with Bilkent University, Ankara, Turkey. His research interests are in the areas of distributed computing, computer networks, and theoretical computer science.