A greedy community-mining algorithm based on clustering coefficient

Aurel Cami and Narsingh Deo
School of Computer Science
University of Central Florida
Orlando, FL 32816-2632
{acami, deo}@cs.ucf.edu

A community in a large, real-life network, such as the World Wide Web (Web), has been broadly defined as a group of nodes that are densely linked with each other, while being sparsely linked with the rest of the nodes. In the last 2-3 years, community-mining in such networks has emerged as a problem of great practical significance. This problem has been framed in at least two different versions: (1) Partitioning into communities refers to partitioning a given network into subsets of nodes, each forming a community; and (2) Seed-growth refers to finding the community to which a given set of “seed” nodes belongs. While several algorithms—employing techniques ranging from hierarchical clustering to spectral partitioning and network flows—have been put forward for the former version, relatively little attention has been devoted to the latter. In this paper we propose a greedy, best-first algorithm for the seed-growth version of community-mining. Beginning with a set of seed nodes, this algorithm grows a community by repeatedly selecting some nodes from community’s neighborhood and placing them in the community. At each step, the algorithm uses clustering coefficient to decide which nodes from the neighborhood are to be included in the community. Our experimental results on both computer-generated and real-world networks indicate that this method may be effective in mining the communities of large networks.

Keywords: community-mining, greedy best-first search, clustering coefficient.

1. Introduction

A variety of natural and man-made systems can be viewed as large networks consisting of nodes and links between them. Important examples of such networks include the Web, the Internet, infrastructure networks, social networks, biological networks, ecological networks, etc. The ever-increasing importance of networks in our lives, has recently motivated a remarkably cross-disciplinary research effort aiming to discover the fundamental properties and the function of networks arising in nature and society. These networks are generally huge,
dynamic and grow without any centralized control. In this paper, we use the term “web-like” to refer to such large, dynamic, and random real-life networks.

Until recently, web-like networks were often modeled by the Erdős-Rényi (ER) classical random graphs [ER59, ER60]. The current revival of interest in studying web-like networks originated a few years ago with some empirical observations which clearly established that such networks differ substantially from the ER random graphs. Most notably, for many web-like networks the proportion $P(k)$ of nodes with degree $k$ decreases as a scale-free power-law, whereas, in the ER model, it is approximately a Poisson distribution. Another marked difference between web-like networks and ER random graphs, is that web-like networks exhibit a significantly greater degree of clustering (see e.g., the recent survey [New03]).

In addition to the just-mentioned observations, recent studies have revealed another interesting property of web-like networks. Many such networks have a locally dense and globally sparse structure consisting of tightly-knit groups of nodes plus few additional inter-group links. Moreover, certain small subgraphs have been determined to occur in web-like networks at numbers that are significantly larger than those in ER random graphs. These properties have been observed in a variety of networks, such as the Web [KRRT99, DKMR01], e-mail networks [GDDG03], science citation networks [ADDG04], biological networks [GN02, MSIK02], etc. The convergence of these empirical findings, has led many researchers to believe that the emergence of such cohesive groups of nodes is an intrinsic property of networks that reflects the processes guiding the evolution of these networks. For example, in neural networks such tightly-knit groups might represent evolved computational units, while in the Web they might represent pages that have a common theme such as the topic of their content, geographical location, etc.

The notion of community has emerged in an effort to formalize these observations. In the broadest sense, a community in a web-like network is a collection of nodes that are densely linked with each other, while being sparsely connected with the rest of the nodes. A question that arises immediately is: what is the best way to precisely define a community? So far this question remains far from being conclusively answered. One may classify the various approaches that have been followed, into two groups. A few authors have proposed rigorous graph-theoretic definitions of community. For example, Kumar et al. [KRRT99] defined a “cyber-community” essentially as a complete bipartite graph, whereas Flake et al. [FLG00] and Radichi et al. [RCCL04] defined a “Web-community” as a set of nodes $S$, such that each node of $S$ has at least as many neighbors inside $S$ as it has outside it. However, many authors [Kle99, GKR98, GN04, BB05] have assumed an algorithmic view of community: rather than

---

1 To be accurate, [KKRT99] defined “cyber-community” as a bipartite core, which is a group of nodes inducing a graph that consists of a complete bipartite graph plus zero or more edges.
formally defining a community, these authors have proposed algorithms that
embed some intuitive notions about communities and then consider the output of
such algorithms as communities. In this paper, we follow the second approach.

Motivation for the community-mining problem

Although the algorithm proposed in this paper might be useful for analyzing a
variety of web-like networks, our main intention is to contribute to the
automated discovery of Web communities. Several scaling and algorithmic
problems being faced today by Web-related applications such as crawlers and
search engines [Hen03, DG03], could potentially be attacked by exploiting
community-mining algorithms, as outlined next:

Increasing the coverage and maintaining the currency of search engine
indices: It has been known for some time that search engines cover only a
fraction of the Web. For instance, in 2000, no search engine covered more than
16% of the Web and the top 11 search engines combined covered about 50% of
the Web [LG00]. Furthermore, many experts agree [CBD99, DG03] that
exhaustive crawling is becoming increasingly unattainable due to the huge size
and dynamic content of the Web. A potential solution to this problem is the
design of focused (or, topical) crawlers [CBD99, DCLG00, MPS04] which
selectively seek out pages that are relevant to a pre-defined topic. Of course,
underlying such a focused crawler there must be an algorithm that directs the
crawl towards the valuable pages, i.e., an algorithm for mining Web
communities.

Reducing the number and increasing the relevance of hyperlinks returned to
a user query: A user searching the Web can be overwhelmed by thousands of
results returned by a search engine. Besides that, queries are often prone to
ambiguity, and sometimes only few of the returned results are relevant. The
PageRank algorithm [PBMW98] took a first step to remedy these issues by
assigning a prestige value to each website and sorting the responses by the
prestige value before returning them. Obviously, more needs to be done. For
instance, if the search engine could group the responses along the lines of
different topics\(^2\), then the user could quickly jump to the desired specific topic.
This application calls for methods to cluster the Web into communities.

Besides these two important applications of Web-community mining, there
are numerous others such as, automatic re-population of topic taxonomies with
newer and more relevant pages [CDAR98], Web filtering (e.g., identification of
hate or pornographic websites) [DVGB03], selective advertising [RC02],
assisting search engines in handling Web spamming [FMN04], etc.

The rest of the paper is organized as follows. In Section 2, we overview
various community-mining algorithms proposed so far. We continue in Section
3 with a discussion of clustering coefficient and in Section 4 with a description

\(^2\) A prototype search-engine which does that can be found at http://clusty.com.
of our community-mining algorithm and its time complexity. In Section 5, we present some experimental results that illustrate the performance of this algorithm and finally in Section 6 we discuss some future work ideas.

2. Related Work

Recently, several community-mining algorithms spanning a wide spectrum of techniques have been proposed. Without claiming to be comprehensive, we have classified these algorithms into three main groups: algorithms based on (a) clustering; (b) spectral; and (c) network-flow techniques.

Algorithms based on clustering

The algorithms in this group have generally employed the technique of hierarchical clustering, which is essentially based on the computation of certain measures of “similarity” between distinct nodes and may be performed in either a bottom-up or a top-down fashion. An agglomerative hierarchical clustering algorithm begins with each node in a separate cluster, and then iteratively, in a bottom-up fashion, pulls together the two clusters that are the most similar. Two measures of similarity borrowed from the field of bibliometrics—bibliographic and co-citation coupling—have been used frequently in community-mining, agglomerative, clustering algorithms [HKKS04][BD05]. In contrast with agglomerative clustering, a divisive clustering algorithm follows a top-down approach to iteratively identify pairs of adjacent nodes that are most “dissimilar”, and remove the edge(s) between them. Usually, the iteration ends when the graph breaks into disconnected components, which then represent the desired clusters. A measure of similarity proposed by Girvan and Newman [GN02], called the “edge betweenness”—the number of shortest paths passing through an edge—has gained some popularity with divisive clustering methods. Another measure, called the “edge clustering coefficient”—the analog of the vertex clustering coefficient (see Section 3)—was proposed by Radichi et al. [RCCL04]. Castellano et al. [CCLP04] combined a divisive clustering method with a formal definition of a community as a group of nodes where each member has more neighbors inside the group than outside it.

In spite of producing groups of nodes that are densely linked with each other while being sparsely linked with the rest of the nodes, clustering algorithms have considerable time demands which limits their application to networks of moderate size.

Algorithms based on spectral techniques

In order to discover the communities contained in a graph, such algorithms exploit the spectra of that graph—i.e., the set of eigenvalues of certain matrices associated with the graph. The earliest applications of spectral techniques for mining communities are Kleinberg’s HITS (Hypertext Induced Topic Search)
algorithm [Kle99] and its variations [BH98, DH99]. This algorithm takes as input a subset of the Web graph and generates two scores for each node in this graph: the “authority” score and the “hub” score. Conceptually, authorities are pages that are considered authoritative on a subject, while hubs are pages that link to authorities. The two scores generated by HITS characterize the degree to which a page satisfies the respective property and they are related to the eigenvectors of the matrixes $AA'$ and $A'A$, where $A$ is the adjacency matrix of the graph under consideration. Capocci et al. [CSCC04] and Donetti and Munoz [DM04] have also proposed community-mining algorithms based on spectral techniques.

The advantage of spectral methods is that they are elegant and often produce good results. However, these methods, too, are not applicable to very large networks due to their time complexity (at least quadratic in the order of the graph).

**Algorithms based on network-flows**

The max-flow/min-cut algorithm lies at the heart of some recent methods for mining Web communities proposed by Flake et al. [FLG00, FLGC02]. The basic algorithm proposed by these authors, aims to discover the community to which a given set of web pages belongs. This problem is cast into an s-t network flow problem by first constructing a graph $G$ that contains all the neighborhoods of the seed pages up to a certain depth, and then adding two artificial nodes: a source node that links to each seed page with an edge of infinite capacity, and a sink node which links to every node of the graph with an edge of capacity $\alpha$—a parameter of the algorithm. The community containing the seed pages is then obtained by running a modified version of the max-flow/min-cut algorithm.

A few additional algorithms that do not fall under any of the three preceding categories have also been proposed. For example, a greedy algorithm that optimizes “modularity”—a measure of the quality of a partition into communities—was given in [New04] and alternate strategies for optimizing the same measure were proposed in [CNM04]. Greco et al. [GGZ04] modeled web communities as bipartite graphs (where hubs contain links to authorities) and then derived the expected evolution of such communities analytically. Based on the results of their analytical work, these authors developed an algorithm for mining communities. Finally, Bagrow and Boltt [BB05] proposed a local, greedy, community-mining algorithm based on degree.

Before describing our community-mining algorithm, we continue next with a discussion of clustering coefficient.

### 3. Clustering Coefficient

Clustering coefficient [WS98] is a graph parameter that measures the degree to which the neighbors of a vertex are neighbors themselves. This parameter has attracted a lot of attention recently, in part due to the surprising discovery that
many real-world networks exhibit a clustering coefficient much higher than that of ER random graphs of the same order (see e.g., [New03]).

Let \( G = (V, E) \) be a graph and \( v \) any vertex in \( G \) of degree 2 or more. Denote by \( o(v) \) the number of opposite edges of \( v \) (i.e., edges of the form \((u, w)\) where \( u \) and \( w \) are two distinct neighbors of \( v \)) and by \( p(v) \) the number of potential opposite edges of \( v \) (that is, \( p(v) = d_v (d_v - 1)/2 \)). Then, the clustering coefficient of vertex \( v \) is given by \( C(v) = o(v)/p(v) \). Note that, the preceding definition is only valid for vertices of degree at least 2. The clustering coefficient, \( C(G) \), of the whole graph is defined as \( C(G) = \sum_{v \in V'} C(v) / |V'| \), where \( V' \) is the set of vertices with degree 2 or more.

A related graph parameter, called transitivity and usually denoted by \( T(G) \), is defined as \( T(G) = 3 \sum_{v \in V'} C(v) / |V'| \), where \( V' \) is the set of vertices with degree 2 or more.

For any graph \( G \), both \( C(G) \) and \( T(G) \) lie between 0 and 1. Furthermore, these two quantities agree on certain classes of graphs. For instance, if (i) \( G \) is regular, or (ii) all nodes of \( G \) have the same clustering coefficient, then it may be easily shown that \( C(G) = T(G) \). In particular, for the complete graph \( C(K_n) = T(K_n) = 1 \), while for the complete bipartite graph \( C(K_{m,n}) = T(K_{m,n}) = 0 \). However, \( C \) and \( T \) differ for many graphs. In fact, these two parameters can be maximally different and neither of them dominates the other [SW04].

4. A Greedy Algorithm for Community Mining

The community-mining algorithm described in this section was designed with several considerations in mind: (a) First, our target application is focused crawling in the Web [CBD99, DCLG00, MPS04]. In order for the algorithm to be suitable for such an application, it has to begin with a small set of seeds and then expand by locally searching their neighborhood; (b) Second, it is aimed to discover a group of nodes that have high density of links among them while having low density of links with the rest of the network. Under these considerations, clustering coefficient seemed a reasonable parameter to guide the search because: (i) a group of nodes having a large clustering coefficients must necessarily have a high density of links (for example, the null graph \( N_n \) has a clustering coefficient of 0, while the complete graph \( K_n \) has a clustering coefficient of 1); and (ii) there is evidence [EM02] that regions of Web that have high clustering coefficients consist of pages that have a common topic. It remained to explore the extent to which it is possible to discover highly-
clustered groups of nodes via a greedy search strategy that favors nodes that have high clustering coefficient with respect to the community nodes. This is discussed in the remainder of the paper.

The pseudo-code for our algorithm, named \textit{FindCommunity}, is shown below. This algorithm takes as inputs a graph $G$, a set $S$ of seed nodes and a threshold $\alpha$. Although many web-like networks are directed, here for simplicity we assume that $G$ is an undirected graph. Beginning with $C = S$, \textit{FindCommunity} grows the community $C$ by repeatedly searching for “valuable” nodes in the neighborhood of $C$.

\begin{algorithm}
\caption{FindCommunity}
\begin{algorithmic}
\Input $G = (V, E)$: an undirected graph
\State $S$: the set of “seed” vertices
\State $\alpha$: threshold
\Output $C$: a set of nodes representing a community that contains $S$
\State $C = S$
\State $N_1 = \text{First neighborhood of } C$
\State $N_2 = \text{Second neighborhood of } C$
\Repeat
\State $I, L = \text{FilterNeighborhood} (G, C, N_1, N_2)$
\State $C = C \cup I$
\State $N_1 = \{\text{First neighborhood of } I\} \cap N_2$
\State $N_2 = \{\text{First neighborhood of } N_1\} - C$
\Until ($N_1$ or $N_2$ become very small)
\Return $C$
\end{algorithmic}
\end{algorithm}

\begin{algorithm}
\caption{FilterNeighborhood}
\begin{algorithmic}
\Input $G = (V, E)$: a graph
\State $C$: community obtained so far
\State $N_1$: the first neighborhood of $C$
\State $N_2$: the second neighborhood of $C$
\State $\alpha$: threshold
\Output $I$: a subset of $N_i$; internal nodes to be included in $C$
\State $L$: complement of $I$ in $N_i$; leaf nodes, not expanded any more
\State $I = \emptyset$, $L = \emptyset$
\ForAll nodes $v$ in $N_i$
\State $(C_0(v), C_1(v), C_2(v)) = \text{ComputeClustCoeff}(G, v, C, N_1, N_2)$
\If ($C_0(v) - C_2(v) \geq \alpha$) \Or ($C_1(v) - C_2(v) \geq \alpha$)
\State $I = I \cup \{v\}$
\Else
\State $L = L \cup \{v\}$
\EndIf
\EndFor
\Return $I$, $L$
\end{algorithmic}
\end{algorithm}
These “valuable” nodes are found by calling the procedure \( \text{FilterNeighborhood} \), which partitions the nodes of the first neighborhood \( N_1 \) of \( C \) into two subsets: the set \( I \), which consists of the nodes to be included in the community (called \textit{internal} nodes), and the set \( L \), which consists of the non-community (or, \textit{leaf}) nodes. The most critical (and time-consuming) computation of procedure \( \text{FilterNeighborhood} \) lies in Step 2, namely, in the call to the procedure \( \text{ComputeClustCoeff} \). This procedure is called for each node \( v \) of the first neighborhood \( N_1 \) and returns three scores: (i) \( C_0(v) \)—the clustering coefficient of node \( v \) with respect to the subgraph induced by \( C+v \); (ii) \( C_1(v) \)—the clustering coefficient of node \( v \) with respect to the subgraph induced by \( N_1+v \); and (iii) \( C_2(v) \)—the clustering coefficient of node \( v \) with respect to the subgraph induced by \( N_2+v \). Here we have assumed that clustering coefficient of a vertex \( v \) is computed by a brute-force method, \textit{i.e.}, by counting the number of edges between the neighbors of \( v \) (other options, such as randomized approximations are also possible but are not considered in this paper).

The decision about whether \( v \) will be an internal or a leaf node is done in Step 3 of procedure \( \text{FilterNeighborhood} \). This decision is simple: if node \( v \) is more tightly clustered to the nodes of \( C \) or \( N_1 \) than it is clustered to the nodes of \( N_2 \), by a value that is greater than the threshold \( \alpha \), then \( v \) is placed in the set \( I \), otherwise it is placed in the set \( L \). In all the experimental results presented later, the parameter \( \alpha \) was fixed at \( \alpha = 0.05 \)—a value which was empirically found to yield good results. To keep the pseudo-code simple we have omitted the treatment of vertices with degree less than 2 (for which the clustering coefficient is meaningless). These vertices are handled following the same reasoning as in Step 3 of \( \text{FilterNeighborhood} \). For example, if a node \( w \) of \( N_1 \) has a single neighbor in \( C \) and no neighbor in \( N_2 \) then it is placed in \( I \). The remaining cases are handled similarly.

\( \text{FindCommunity} \) algorithm runs until the first or second neighborhoods become very small. The stopping criteria used in our experiments was \( |N_1| < 1 \) or \( |N_2| < 2 \).

As a first illustration, in Figure 1, we have shown how the \( \text{FindCommunity} \) algorithm performs in a trivial case: two complete graphs \( K_{10} \) joined by an additional edge. We selected two random seed nodes from the first \( K_{10} \) (nodes 1 and 6, shown in white in Figure 1(a)). Thus, initially \( N_1 = \{2, 3, 4, 5, 7, 8, 9, 10\} \) (nodes shown in grey color in Figure 1(a)) while \( N_2 \) consists of all the nodes of the second \( K_{10} \) (shown in black color in Figure 1(a)). It one step (\textit{i.e.}, one execution of Steps 4-7) of \( \text{FindCommunity} \), all the nodes of the first \( K_{10} \) are classified as community nodes (shown in white in Figure 1(b)).

\textit{Time complexity}

It is difficult to obtain an exact expression for the time complexity of \( \text{FindCommunity} \) in terms of the \textit{order} and \textit{size} of the graph \( G \). However, assuming that the number of nodes visited by this algorithm
Figure 1. A graph consisting of two complete graphs $K_{10}$ and an additional edge that links them together. (a) initial configuration (b) after one step of the FindCommunity algorithm.

5. Experimental Results

We implemented the above algorithm and tested its performance on several networks. The implementation was done in C++ using LEDA\(^3\) software package, and testing was carried out on a Linux box. The results of our experiments are shown next.

Zachary’s karate club network

The first network we used to test the FindCommunity algorithm, is the Zachary’s karate club network [Zac77]—a real-life network with 34 nodes and two communities, which has become a frequently-used benchmark for community-mining algorithms. The nodes of this network represent the members of a karate club at an American university, while the edges represent their social interactions. Zachary’s network is known to consist of two different communities of nodes, each corresponding to the club members that sided with one of the two club leaders during a dispute (see e.g., [GN03] for a more detailed description of this network). Figure 2 shows the performance of our

\[^3\] Available from http://www.algorithmic-solutions.com/enleda.htm
algorithm in discovering each of the two communities of this network, beginning in each case from a single seed node.

![Figure 2. Discovering the two communities of Zachary's karate club network: (a) Node 1 (white) is chosen as the seed for the first community; (b) The white nodes represent the community found by the algorithm; (c) Node 34 (white) is chosen as the seed for the second community; (d) The white nodes represent the community found by the algorithm.](image)

In the case of discovering the first community with node 1 as seed, all except two nodes were classified correctly (Figure 2(a, b)). In the case of discovering the second community with node 34 as seed all but three nodes were classified correctly (Figure 2(c, d)).

**Random graphs with known community structure**

Next, we tested the algorithm extensively on a family of random graphs with known community structure. This family of random graphs has also been used frequently to test community-mining algorithms, e.g., [NG03, BB05]. A random graph from this family is characterized by the following parameters: (1) $n$—number of nodes; (2) $c$—number of communities; (3) $d_{in}$—the expected in-degree of a vertex; (3) $d_{out}$—the expected out-degree of a vertex (here the subscripts $in$ and $out$ stand for in-community and out-of-community).
following simple algorithm may be used to generate such a random graph $G(n, c, d_{in}, d_{out})$:

1. Let $CommunitySize = n / c$
2. Let $p_{in} = d_{in} / CommunitySize$, $p_{out} = d_{out} / (n - CommunitySize)$
3. Partition the set of nodes into $c$ communities of equal size (assuming $c$ divides $n$ evenly)
4. For each pair of nodes $(u, v)$, independently do the following:
   a) if $u$ and $v$ belong to the same community, join them with an edge with probability $p_{in}$
   b) otherwise, join them with an edge with probability $p_{out}$

We focused on three main questions related to the performance of FindCommunity algorithm on this family of random graphs: (i) How does the accuracy of the algorithm—measured as the fraction of nodes classified correctly—change while the ratio $d_{in}/d_{out}$ increases? (ii) What is the impact of the size of the set of seed nodes on the accuracy of the algorithm? and (iii) How robust is the algorithm to different sets of seed nodes?

To investigate these three questions we carried out a number of experiments. The following scenario is common to all them: While keeping the rest of the parameters fixed, vary the parameter of interest and for each case generate a random graph having the desired parameters; Randomly choose a fraction of the nodes of the first community of this random graph as seed nodes; Run FindCommunity algorithm; Compute the fraction of nodes classified correctly by this algorithm (a node from the first community is considered correctly classified if it is classified as community-node; likewise for non-community nodes).

**Accuracy versus $d_{in}/d_{out}$**

Figure 3, shows how the accuracy of the algorithm changes with the ratio $d_{in}/d_{out}$ for two random graphs with 16,384 nodes and with 4 and 32 communities, respectively. In this experiment, $d_{in}$ is kept fixed at 32 while $d_{out}$ is varied from 4 to 28 (i.e., the ratio $d_{in}/d_{out}$ varied from 0.125 to 0.875). In both cases, the set of seed nodes, consisted of 5% of the nodes of the first community, selected uniformly at random. As seen in Figure 3, the fraction of correctly classified nodes changed from nearly 0.9 (when $d_{in}$ equals 4) to nearly 0.7 (when $d_{in}$ equals 28).

**Size of the set of seed nodes**

Figure 4, shows the impact of the relative size of the set of seed nodes on the accuracy of the algorithm. In this case, two random graphs with 16,438 nodes and with 8 and 32 communities, respectively, were generated. In both cases $d_{in} = 32$ while $d_{out} = 8$. 
Figure 3. Fraction of correctly classified nodes versus the ratio $d_{in}/d_{out}$.

The size of the set of seed nodes was varied from 2% of community size, to 20% of the community size. In the case of the graph with 32 communities, the fraction of correctly classified nodes changed from nearly 90% to almost 99%. In the other case, when only 8 communities are present, this fraction was smaller than in the first case, but always greater than 83%.

Figure 4. Fraction of correctly classified nodes versus the relative size of the set of seed nodes.

Robustness to different seeds

Figure 5. illustrates the robustness of the algorithm to different sets of seeds. In this experiment, we generated a random graph with 16,384 nodes, with 8 communities, and with $d_{in} = 32$, $d_{out} = 8$. The FindCommunity algorithm was executed ten times with this graph as input. For each execution, a new set $S$ was
generated by selecting 5% of the nodes of the first community uniformly at random. As seen in Figure 5, the fraction of nodes classified correctly varied very little with the set of seed nodes—it was always between 95% and 96.5%.

![Figure 5. Robustness of the algorithm to different sets of seed nodes.](image)

In summary, FindCommunity algorithm achieves good accuracy in mining the community to which a given set of seed nodes belongs. The fraction of nodes classified correctly was generally above 80% and often above 90% when the algorithm was tested on some random graphs with known community structure. Furthermore, the algorithm achieves good accuracy with only a small fraction (about 1%) of community nodes as seeds and the performance varies little with the set of seed nodes.

6. Concluding Remarks

In this paper, we proposed a greedy, best-first algorithm for discovering the community to which a given set of nodes belongs. This algorithm achieves good accuracy when tested on some real and computer-generated networks. However, several issues need to be addressed before this algorithm can be usefully applied in practice. First, more extensive testing needs to be done especially with data from real networks such as Web crawls. Second, a rigorous method needs to be developed to evaluate the quality of the communities produced by this algorithm. Third, in order to mine large communities (in the order of tens of thousands of nodes), the speed of the algorithm needs to be improved (without compromising its accuracy). One could consider randomizing the ComputeClusteringCoeff procedure, perhaps in combination with using a generalized clustering coefficient, where not only the first neighborhood is taken into account but also the second, the third, etc. Another option would be to parallelize the algorithm. Investigating these questions as well as applying this algorithm to mine the communities of some web-like networks, remain as topics of our current and future research.
7. References


