THE NATIONAL UNIVERSITY
of SINGAPORE

School of Computing
Computing 1, 13 Computing Drive, Singapore 117417

TRC5/13

Fast Community Detection

Yi Song and Stéphane Bressan

May 2013

T e c h n i c a l R e p o r t

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

OOI Beng Chin
Dean of School

Fast Community Detection

Yi Song and Stéphane Bressan

School of Computing
National University of Singapore
{songyi, steph}@nus.edu.sg

Abstract. We propose an algorithm for detecting communities in networks. The algorithm exploits degree and clustering coefficient of vertices as we hypothesize that these metrics characterize dense connections indicative of a community. Each vertex, independently, seeks the community to which it belongs by visiting its neighbor vertices and choosing its peers on the basis of their degrees and clustering coefficients. The algorithm is intrinsically data parallel. We devise a version for *graphics Processing Unit* (GPU). We empirically evaluate the performance of our method. We measure and compare its efficiency and effectiveness to several state of the art community detection algorithms. Effectiveness is quantified by five metrics, namely, modularity, conductance, internal density, cut ratio and weighted community clustering. Efficiency is measured by the running time. Clearly the opportunity to parallelize our algorithm yields an efficient solution to the community detection problem.

1 Introduction

Communities forms when groups of vertices in the network are connected more to their group peers than to the other vertices in the network. Discovering such groups or communities helps to find efficient ways to distribute or gather information in online social network for example. It is a useful tool in fields such as sociology, biology and marketing. In this paper, we propose an algorithm for efficient detecting good communities.

We model the networks as simple graph $G(V, E)$. V is a set of vertices and E is a set of edges. It is undirected, un-weighted, and has no self-loop. The idea of our method is for each vertex to seek the community to which it belongs by visiting its neighbor vertices. Decisions are made based on the degrees, clustering coefficients of the neighbors and the number of common neighbors. Our method starts from micro perspective, which is different from the previous works such as GN. Considering the size of networks nowadays, we would like our method to have a decent scalability so that it can have the capability to deal with the large networks in a short time. Therefore we intend to have as few as possible the pairs-wise computation among vertices. Instead of finding similar vertices by computing similarities of all pairs of vertices in graph, we suggest to get similar vertices by looking around neighborhood only. Anyway the similar vertices in

the same community are more likely among the neighbors. This significantly reduces the complexity except in the case of complete graphs. In our algorithm, as each vertex independently evaluates its neighborhood and joins a community by following one of its immediate neighbors or stands alone, the algorithm is intrinsically data parallel. We devise a parallel version for *graphics Processing Unit* (GPU) according to the sequential version for CPU.

We empirically evaluate the performance of our algorithm with both real world networks and synthetic networks. We evaluate the quality of communities by metrics from different classes [36], and one recently proposed metric [28]. The metrics include modularity, conductance, internal density, cut ratio, and weighted community clustering (WCC). Those metrics indicate the community quality from different perspectives. We measure the running time. We compare our algorithm with several the state-of-the-art algorithms.

The rest of the paper is organized as follows. Section 2 briefly reviews the related works on graph clustering and community detection. Section 3 presents the algorithm we propose. Section 4 shows the experiment setting, experiment results and results analysis. Finally we conclude in Section 5.

2 Related Work

Graph clustering and community detection methods can be categorized into several classes. Several authors [27] [18][37] [17] [33] use random walks. For example, Pascal and Matthieu [27] used random walk to calculate the similarities which they call distance between each pair of adjacent vertices, and then use Ward's agglomerative hierarchical clustering approach to find communities. Jin et al. [18] proposed an algorithm based on Markov random walk to unfold the communities, and extract them with cutoff criterion in terms of conductance. Dongen [33] propose Markov Clustering which simulates the random walks and calculates them by Markov Chain.

Several authors [25][4][26][15][16] focus on *modularity* which is first proposed by Girvan and Newman [13]. Modularity is invented as the number of edges inside groups minus the expected number in an equivalent graph with edges placed at random. An equivalent graph here means that the graph has the same number of edges and the same degree distribution. For example, Clauset [4] defines a local measurement of community structure called *locally modularity* and proposes an agglomerative algorithm to maximize the *local modularity* of the communities detected. Girvan and Newman [25] propose a divisive method to identify community. The edges with highest betweenness are removed iteratively, which disconnects the graph into communities. The best partition has the highest *modularity*.

Some authors [9][12] use cliques. For example, Du et al. [9] use maximal cliques for community detecting. An algorithm called *ComTector* is proposed. It enumerates all maximal cliques for finding clustering kernel, assigns the rest vertices to closest kernels, and merges fractional communities. Palla et al. [12]

design the clique percolation method (CMP) which finds all cliques of size k. Communities are connected union of k-cliques.

The authors of [30][5][1] detect community in an agglomerative way. Ahn et al. [1] define clusters as sets of edges. Their method group edges with an agglomerative hierarchical clustering technique. Clauset et al. [5] propose a greedy hierarchical agglomerative algorithm. It starts from each vertex being a community and then joins two communities at each iteration. The two communities are selected based on the idea of maximizing modularity increment. They use dendrogram to represent the whole process.

Besides those ideas, Jierui and Boleslaw [34] propose speaker-listener label propagation algorithm for overlapping community detection. Zhang et al. [38] propose a method that combines spectral mapping, fuzzy clustering and the optimization of a quality function. Yan and Gregory [35] propose an optimization algorithm for existing community detection algorithms. Pairwise vertex similarities are measured beforehand, and existing algorithms are applied on the graph with the vertex similarities as edge weights. Rosvall and Bergstrom [29] use information theoretic approach to detect community in weighted and directed network.

Some methods, such as those presented in [2][3][14][19][6], detect community locally, which is closer to but not the same as ours. For example, Baumes et al. [2][3] propose two heuristics to detect locally dense subgraphs as communities. Two different subgraphs with significant overlap can be both locally optimal and thus they are overlapping communities. The first heuristic finds disjoined clusters by deleting high-ranking vertices and then adds the deleted vertices to one or more clusters. The second one starts from randomly chosen seeds and then adds or deletes one vertex at a time till the density metric cannot be improved further. Goldberg et al. [14] propose an additional requirement based on [2][3] which requires the community to be a connected sub-graph. so that the algorithm is able to examine the connectivity of the cluster found.

3 Algorithm

We propose an algorithm that delegates the job of finding communities to each vertex. Vertex seeks its community independently. The decisions of which community to join are made based on the degrees and clustering coefficients of neighbors, as well as the number of common immediate neighbors. We hypothesize that the vertices tend to join groups with more connections. In other words, the vertices try to attach themselves to dense structures, structures with more connections among vertices in this structure.

The algorithm starts by calculating degrees and local clustering coefficient for each vertex (line 1). Local clustering coefficient is defined as

$$cc[i] = \frac{e_{jk} : j, k \in V, e_{jk} \in E}{degree[i] * (degree[i] - 1)}$$

It is the ratio between the number of edges between vertices within its neighborhood and the number of edges that could possibly exist between them. It quantifies how close the vertex connects with its neighbors.

Algorithm 1: Fast Community Detection

Input: graph $G(V, E)$ with $|V|$ vertices, $|E|$ edges;
Result: Clusters C_i, $i \in (1, 2, ..., k')$

Compute degree[v] and cc[v], $v \in V$;
for *each* v **do**
  if *degree[v]<degree[v_j]* **then** /* $v_j \in v_{Neighbor}$ */
    g[v] $\leftarrow v_i$, where degree[v_i]=max(degree[v_j]) ;
  else
    g[v] = v;
for *each* v **do**
  if *g[v]* = v *and degree[v]=degree[v_i]* **then**
    if v *and* v_i *has more than half common vertices;*
    then
      g[v]$\leftarrow v_i$, if v_i has smaller id;
  else
    $v_g \leftarrow$ g[v];
    $c1 \leftarrow$ number of common neighbors between v and j;
    $c2 \leftarrow$ number of common neighbors between v and $(v_{Neighbor} \setminus v_g)$;
    if *c1<c2* **then**
      g[v] $\leftarrow v_i$, where degree[v_i]=max(degree[v_j]), $v_j \in (v_{Neighbor} \setminus v_g)$
for *each* v **do**
  if *g[v]* $\neq v$ **then**
    $i \leftarrow$ g[v];
    repeat
      $i \leftarrow$ g[i] ;
    until *g[i]*= i;, find stand alone vertex
    g[v]$\leftarrow$i;
$k \leftarrow$ different numbers in g[v];
for i *from 1* **to** k **do**
  for $v \in C_i$ **do**
    find the cluster C_j where v has the maximum number of immediate neighbors;
    if $i \neq j$ **then**
      Cluster v into C_j;
Return C_i,$i \in (1, 2, ..., k')$;

In the second step, each vertex look around its immediate neighbors. If the degree of the vertex, for example vertex v, is the largest among its immediate neighbors, vertex v stands alone and does not follow other vertices. If the degree of vertex v is not the largest among its immediate neighbors and itself, vertex v follows the neighbor with the largest degree among v's immediate neighbors

(line 2-6). If more than one vertex among the immediate neighbors have the largest degree, then vertex v follows the one with the largest clustering coefficient compared to other neighbors.

In the second round, each vertex adjusts their decisions (line 7-17). If the standing-alone vertex v has neighbors with the same degree, check the number of common neighbors of vertex v and v's neighbor that has the same degree. If there are enough common neighbors, these two vertices are suggested to be in the same community. If the vertex v does not stand alone but follows some neighbor, we check the number of common neighbors vertex v has with the vertex that it follows, and the number of common neighbors it has with the other neighbors. If vertex v has more common neighbors with its other neighbors than the one it follows, then vertex v turn to the vertex with the second largest degree in the neighborhood or stands alone if itself has the second largest degree.

The third round, each vertex finalizes the community which it desired to join (line 18-24). If the vertex that vertex v follows is also following vertex v_i, than vertex v also turn to vertex v_i. In the end, each vertex follows a vertex that stands alone. With all the other vertices that follow this vertex, they form a community.

After each vertex chooses its community (line 25), we post-process the memberships to refine the communities (line 26-30). If any vertex has more connections outside the community than inside the community, it changes its membership. This refine process may change the number of communities from the last step.

The only input of the algorithm is the graph itself. In experiments the graph is inputted in the form of edge list as input. No pre-defined number of communities is needed. The output is the communities.

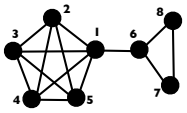


Fig. 1. Example

Figure 1 shows a graph with 8 vertices and 14 edges. After the first round, vertex 2, 3, 4, 5, 6 all follow vertex 1 (g[1]=1, g[2]=1, g[3]=1, g[4]=1, g[5]=1, g[6]=1), while vertex 7 and 8 follow vertex 6 (g[7]=6, g[8]=6). In the second round for each vertex, the status of vertex 1 is unchanged. The status of vertex 2, 3, 4, 5 is also unchanged, because they more common neighbors with vertex 1 that they follow than with other vertices ({vertex 2, 3, 4, 5}\themselves), vertex 7 and 8 still follow 6, while vertex 6 changes to be standing alone instead of following vertex 1 because vertex 6 has more common neighbors with 7 and 8 than with vertex 1. No more changes happen in the third round and the refinement, and thus final result is that we find two communities: one community is labelled by

vertex 1, and has vertex 1, 2, 3, 4, 5; the other community is labelled by vertex 6, and has vertex 6, 7, 8.

As the algorithm is parallelizable, we devise a parallel version. In this parallel version, what the vertices do in the first round is parallelized as well as what the vertices do in the second round. In other words, in the first round the vertices look for the vertex with the largest degree in the neighborhood at the same time. In the second round, the each vertex adjusts their decisions at the same time. The rest of the algorithm is still sequential.

The time complexity for calculating clustering coefficient is $\mathcal{O}(n * d^2)$, where n is the number of vertices and d is the maximum degree of vertices in graph. The complexity for the first round is $\mathcal{O}(n * d)$. The complexity for the second round is $\mathcal{O}(n * d^2)$. The complexity for the third round is $\mathcal{O}(n^2)$. The complexity for the refinement is $\mathcal{O}(n * d^2)$. Therefore the time complexity for the whole algorithm is $\mathcal{O}(n * d^2 + n^2)$. For the parallel version, The complexity for the first round is $\mathcal{O}(d)$. The complexity for the second round is $\mathcal{O}(d^2)$. The rest is the same as that of the sequential version. Thus the time complexity for the whole parallel algorithm is $\mathcal{O}(d^2 + n^2)$.

4 Experiment

We conduct experiments on both synthetic graphs and real world data including three benchmark graphs for community detection algorithm. We ran the sequential algorithms on an 2.83GHz Inter Core, 2 Quad CPU machine with 2GB of main memory under Windows 8 OS. The algorithms are implemented in C. The parallel algorithm ran on the same machine with a GeForce GTX 560 Ti graphics card having 2048 MB of global memory, 8 multiprocessor and 48 CUDA cores per multiprocessor. All algorithms were implemented in Visual C++ 10.0. This parallel algorithm are implemented using the application programming interface CUDA from the C language. CUDA [7], the C language Compute Unified Device Architecture, is provided by NVIDIA and works on NVIDIA graphic cards. The CUDA programming model consists of a sequential host code combined with a parallel kernel code.

We compare our algorithm with three state-of-the-art algorithms: *InfoMap* [29], *WalkTrap* [27] and Girvan and Newman (*GN*)[13][25]. *InfoMap* is on information theory. *Walktrap* is based on random walk. *InfoMap* has been empirically shown to have better performance compared with other algorithms for community detection [11].

4.1 Dataset

We generate a batch of benchmark graphs [20] with known community structure, number of vertices, the average degree, maximum degree, minimum and maximum size of micro and macro community due to the hierarchical structure, and fraction of edges between vertices belonging to same or different communities.

In our experiments, we generate graphs with 2000 vertices and different average degrees while the other parameter are the same. They have no overlapping communities.

The real-world benchmark graphs we use are listed as followings. Among them Zachary's Karate Club data, American College Football data and Dolphin network are widely used for evaluating community detection algorithms.

Karate Club data is a social network of karate club members studied by the sociologist Wayne Zachary. The network has 34 members (vertices) and they separated into two different groups due to a controversy between one of the instructors and administrator of the club.

American College Football data is a network with 115 teams (vertices) which are separated into 12 conferences. An edge exists between two vertices if there is match between two teams. More games happen among teams within the same conference than teams from different conferences.

Dolphin Network is collected by David Lusseaua [23]. The network represents frequent associations between 62 dolphins (vertices) in a community living off Doubtful Sound, New Zealand.

Email-URV data is collected by Guimer et al. [8]. The network contains user-to-user (address- to-address) links from the network of e-mail interchanges among faculty and graduate students at Rovira i Virgili University of Tarragona, Spain. It's available on Alex Arenas Website [10].

Arxiv HEP-PH collected by Leskovec et al. [22], is a collaboration network contains scientific collaborations between authors who submitted papers to High Energy Physics. It's available on SNAP website [31].

Wiki-Vote, collected by Leskove et al. [21], contains user-to-user (who-vote-whom) links from Wikipedia network. It's available on SNAP website [31]. Each vertex represents a user. An edge is created from a user to a candidate if a user votes for Wikipedia admin candidates.

Email-Enron data set contains user-to-user (address-to-address) links. It was made public by the Federal Energy Regulatory Commission during its investigations. We obtained it from [31]. Each vertex represents an email address. An edge exists between vertex i and vertex j if address i sends at least one email message to address j.

Epinions data set contains user-to-user (who-trust-whom) links from Epinions network. It was collected by Epinions staff P. Massa. We obtained it from *trustlet* website [32][24]. Each vertex represents a user. An edge corresponds to a trust or distrust statement from one user to another user.

We extract the largest component of the networks that have more than one component. The number of vertices and the number of edges of each data are listed in Table 1

4.2 Metrics

We use five metrics to qualify the communities: modularity, conductance, internal density, cut ratio and weighted community clustering. Modularity, conductance, internal density and cut ratio are selected from four classes of metrics for

Table 1. statistics of datasets

	Number of Vertices	Number of edges
Karate Club	34	78
Dolphin	62	159
American College Football	115	610
Email-URV	1,133	5451
Wiki-Vote	7,066	100,736
Arxiv HEP-PH	11,204	117,649
Email-Enron	33,696	180,811
Epinions	119,130	704,276

community [36] so that we can eliminate the bias of having only one kind of metric. Weighted community clustering is a recently proposed metric [28].

The **Modularity** [25] is defined as

$$modularity = \frac{1}{2m}\Sigma_{i,j\in V}(A_{ij} - \frac{k_i k_j}{2m})\delta(c_i, c_j)$$

where $A_{ij} = 1$ if i and j are connected, otherwise $A_{ij} = 0$, and $\delta(c_i, c_j) = 1$ if i and j belong to the same cluster, otherwise $\delta(c_i, c_j) = 0$.

The **Conductance** for a set of vertices S is defined as

$$conductance(S) = \frac{c_s}{2m_s + c_s}$$

where $c_s = |(u, v) \in E : u \in S, v \notin S|$. It is the number of edges with one end in the set and the other end outside the set. $m_s = |(u, v) \in E : u \in S, v \in S|$. It is the number of edges in S.

The **Internal Density** for a set of vertices S is defined as

$$InternalDensity(S) = \frac{m_s}{n_s(n_s - 1)/2}$$

where m_s is the same as above. n_s is the number of vertices in S. Internal Density is the internal edge density of S.

The **Cut Ratio** for a set of vertices S is defined as

$$CutRatio(S) = \frac{c_s}{n_s(n - n_s)}$$

Cut Ratio is the fraction of existing edges out of all possible edges having one end outside the cluster.

The Weighted Community Clustering for a is defined as

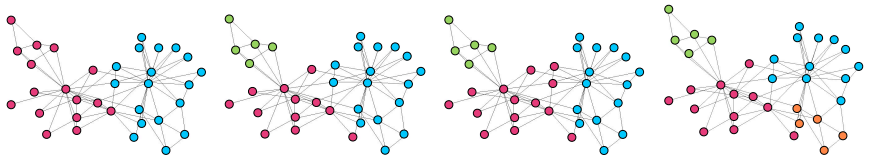
$$WCC(S) = \frac{1}{|S|}\sum_{x\in S} f(x, S)$$

where $f(x, S) = \frac{t(x,S)}{t(x,V)} * \frac{vt(x,V)}{|S\backslash x| + vt(x,V\backslash s)}$ if $t(x, V) \neq 0$; $f(x, S) = 0$ if $t(x, V) = 0$. $t(x, S)$ is the number of triangles that vertex x closes with vertices in S and $vt(x, S)$ is the number of vertices of S that form at least one triangle with x.

In our experiments, we take the average of the conductances of communities found for the conductance of the whole network, and same for the other metrics except modularity.

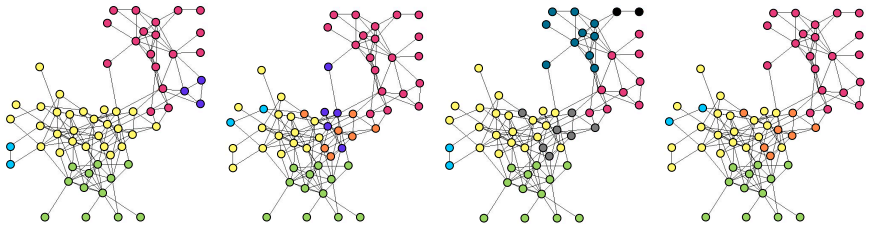
4.3 Experimental Assessment and Analysis

Figure 2 shows the communities found in the Karate Club network by each algorithm. Figure 3 shows the communities found in the Dolphin new network by each algorithm. Vertices in the same color are in the same community.



(a) FCD (b) InfoMap (c) WalkTrap (d) GN

Fig. 2. communities for Karate Club data by different algorithms



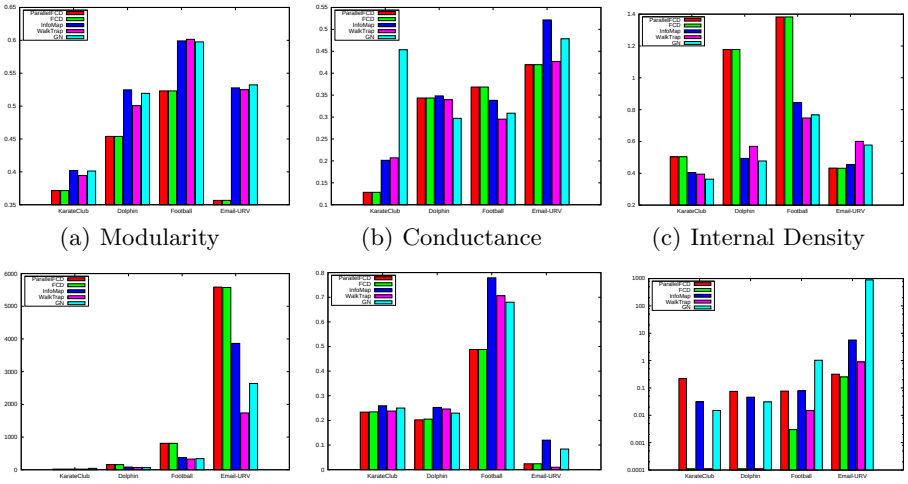
(a) FCD (b) InfoMap (c) WalkTrap (d) GN

Fig. 3. communities for Dolphin data by different algorithms

Figure 4 shows the measurement results on the four real data sets. X-axis is labelled by the names of data sets. Y-axis is the value of metrics. For each data set, the metric values for the community detected by each algorithm are compared. Figure 4 (a) shows that the communities that FCD and ParallelFCD found have lower modularity on these four datasets. However, this does not indicate that our algorithm is not better than the other three algorithms. Figure 2 shows that our algorithm identifies two communities that coincide with the truth that the members of the Karate Club separated into two different groups due to the controversy, and thus the result of our algorithm is actually more reasonable than the other three algorithms even though the modularity values are lower. Figure 4 (b) shows the conductance results. The lower the conductance, the better the communities found. In this case, our algorithm has lowest conductance on two data sets and highest conductance on the other two data sets. Figure 4 (c) shows the internal density results. The higher the internal density, the better the communities found. In this case, our algorithm has highest internal density in three of the four data sets, and lowest in one data set. Figure 4 (d) shows the cut

ratio results. The lower the cut ratio the better the communities found. In this case, our algorithm has lowest cut ratio in one of the four data set, and highest in the other three data sets. Figure 4 (e) shows the weighted community clustering results. The higher the *WCC*, the better the communities found [28]. In this case our algorithm has lower *WCC* in three of the four data sets. Figure 4 (f) shows the running time. For all the four data sets here, *FCD* performs fastest among the algorithms. *ParallelFCD* performs faster than *InfoMap*, *WalkTrap* and *GN* on the Email-URV data.

To sum up the results on these four real data sets, our algorithm, *FCD* and its parallel version, find communities with better values in terms of internal density and conductance, but not with the values of other metrics. However, as we see from the results for Karate Club, the communities detected by our algorithm stay more truthful than the others. In this sense, our algorithm is effective. From the comparison of the running time, *FCD* is obviously more efficient than the others.

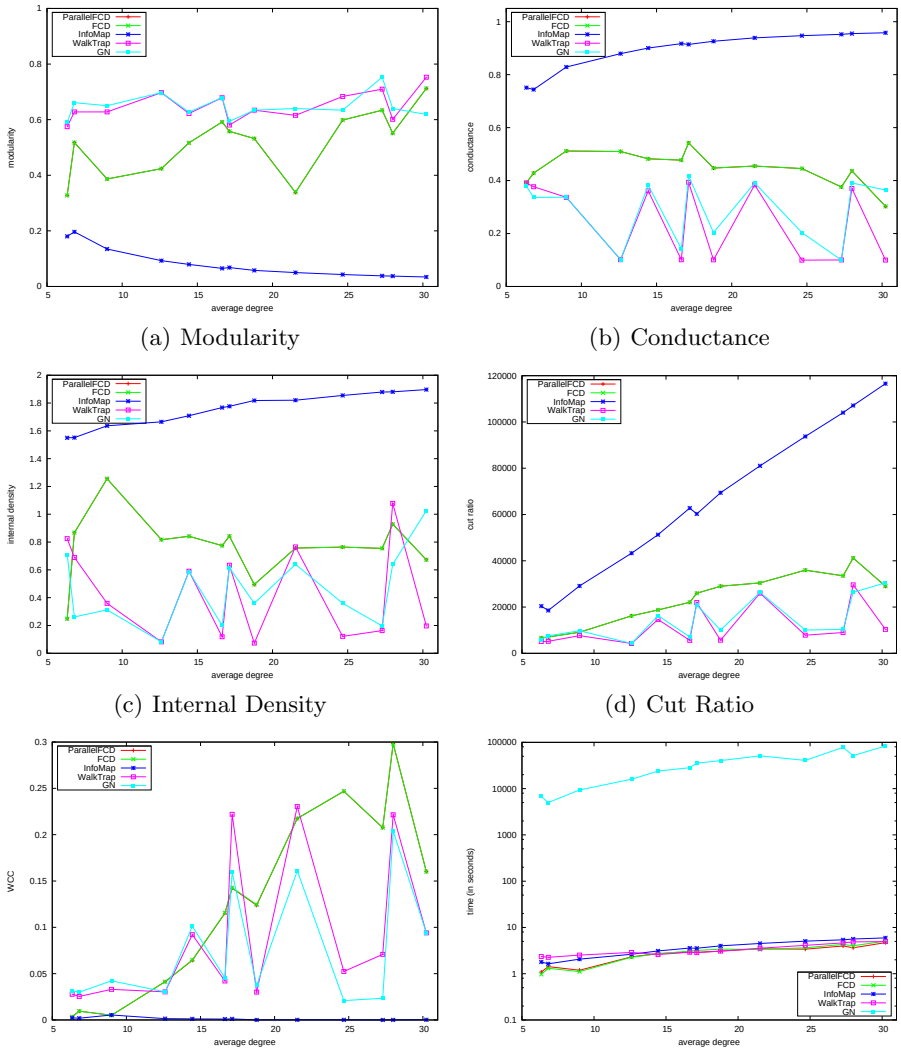


(a) Modularity (b) Conductance (c) Internal Density

(d) Cut Ratio (e) Weighted Community Clustering (f) Running Time

Fig. 4. measurements on real world data

Figure 5 shows the measurement results on the benchmark graphs generated. It shows the value changing as the graphs increase average degrees. X-axis is the average degree of the graphs. Y-axis is the value of metrics. Each dot represents one metric value for the communities detected by one algorithm. Figure 5 (a) shows the modularity results. It shows that *WalkTrap* has the highest modularity in general though in some cases *GN* and *FCD* has the highest modularity, and *FCD* has higher modularity than *InfoMap*. Figure 5 (b) shows the conductance results. It shows that *InfoMap* has the highest conductance and *GN* has the the



(a) Modularity (b) Conductance

(c) Internal Density (d) Cut Ratio

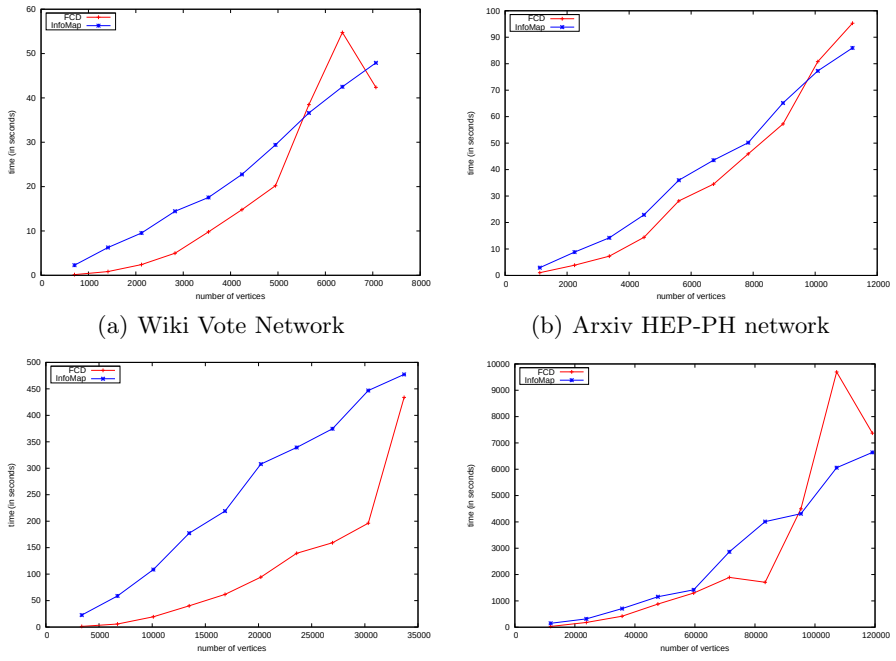
(e) Weighted Community Clustering (f) Running Time

Fig. 5. measurements on synthetic data

lowest. Figure 5 (c) shows the internal density results. It shows that *InfoMap* has the highest internal density, and *GN* has the lowest density. Figure 5 (d) shows the cut ratio results. It shows that *InfoMap* has the highest cut ratio, and *GN* has the lowest. Figure 5 (e) shows the *WCC* results. It shows that *FCD* and *WalkTrap* have higher *WCC*, and *InfoMap* and *GN* has lower *WCC*. Figure 5 (f) shows the running time. *FCD* and *ParallelFCD* are shown to be faster in most cases. *GN* is much slower than *InfoMap*, *WalkTrap* and *FCD*. *ParallelFCD* is not obviously faster than *FCD* due to the data communication between the host *CPU* and device *GPU*

To sum up the results on these synthetic graphs, *FCD* (*ParallelFCD*) performs more stable than *InfoMap* and *GN* in terms of effectiveness. *InfoMap* is the best in terms of internal density but the other three algorithms are better in terms of conductance, cut ratio and *WCC*. *GN* is the best in terms of conductance and cut ratio but the other three algorithms are better in terms of internal density and modularity. Compare with *WalkTrap*, *FCD* is better in most cases in terms of modality and internal density.

FCD performs faster than the other three in general. In other words, *FCD* is more efficient.



(a) Wiki Vote Network

(b) Arxiv HEP-PH network

(c) Email-Enron Network

(d) Epinion Network

Fig. 6. running time for large networks

Another set of experiment tests running time on large networks: Wiki-Vote, Arxiv HEP-PH, Email-Enron, and Epinion network. We sample subgraphs from the networks. Every subgraph contains k percentage vertices of the original networks, where $k = 10, 20, ..., 90$. We run *FCD* and *InfoMap* algorithm on these subgraphs and the original graphs. The running time is recorded. Figure 6 shows the running time changing as the number of vertices of networks increases. Each figure shows the results for one data set. X-axis is the number of vertices. Y-axis is the time measured in seconds. Due to *WalkTrap* and *GN* algorithms' scalability on large graphs, we only compare *InfoMap* and *FCD* algorithm here. The results show that both algorithms are able to work with graphs with more than 10,0000 vertices. For graphs such as Email-Enron with 33,696 vertices, the algorithms are able to finish the task in a few minutes. In most cases *FCD* is faster than *InfoMap*.

5 Conclusions

In this paper we propose a fast community detection algorithm. It initiates each vertex to seek for the community in its neighborhood independently. Each vertex chooses its community and peers based on the knowledge of degrees and clustering coefficients of neighbors and the number of common neighbors. The algorithm is parallelizable and thus we also devise a GPU version of the algorithm for parallel computation. We empirically evaluate the performance of *FCD*, and compare it with *InfoMap*, *WalkTrap* and *GN* algorithms. We find that *FCD* is the fastest among them generally. We assess effectiveness based on the values of modularity, conductance, internal density, cut ratio and weighted community clustering. we find that *FCD* has more stable performance than *InfoMap* and *GN* do, while produces competitive results with *WalkTrap*.

References

1. Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. *NATURE*, 466:761, 2010.
2. J. Baumes, M. K. Goldberg, M. S. Krishnamoorthy, M. Magdon-Ismail, and N. Preston. Finding communities by clustering a graph into overlapping subgraphs. In *IADIS AC*, pages 97–104, 2005.
3. J. Baumes, M. K. Goldberg, and M. Magdon-Ismail. Efficient identification of overlapping communities. In *ISI*, pages 27–36, 2005.
4. A. Clauset. Finding local community structure in networks. *PHYS.REV.E*, 72:026132, 2005.
5. A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *PHYS.REV.E*, 70:066111, 2004.
6. M. Coscia, G. Rossetti, F. Giannotti, and D. Pedreschi. Demon: a local-first discovery method for overlapping communities. *CoRR*, 2012.
7. CUDA-Zone. http://www.nvidia.com/object/what_is_cuda_new.html.
8. L. Danon, A. Diaz-Guilera, F. Giralt, and A. Arenas. Self-similar community structure in a network of human interactions. *Physical Review E*, 68, 2003.

9. N. Du, B. Wu, X. Pei, B. Wang, and L. Xu. Community detection in large-scale social networks. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, WebKDD/SNA-KDD '07, pages 16–25. ACM, 2007.
10. Email-URV. http://deim.urv.cat/ aarenas/data/welcome.htm.
11. S. Fortunato and A. Lancichinetti. Community detection algorithms: a comparative analysis: invited presentation, extended abstract. VALUETOOLS '09, ICST, Brussels, Belgium, Belgium, 2009. ICST.
12. I. F. Gergely Palla, Imre Derenyi and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, June 2005.
13. M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.
14. M. K. Goldberg, S. Kelley, M. Magdon-Ismail, K. Mertsalov, and A. Wallace. Finding overlapping communities in social networks. In *SocialCom/PASSAT*, pages 104–113, 2010.
15. S. Gregory. An algorithm to find overlapping community structure in networks. PKDD 2007, pages 91–102. Springer-Verlag, 2007.
16. S. Gregory. A fast algorithm to find overlapping communities in networks. ECML PKDD '08, pages 408–423. Springer-Verlag, 2008.
17. D. Harel and Y. Koren. On clustering using random walks. In *Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science*, London, UK, UK, 2001. Springer-Verlag.
18. D. Jin, B. Yang, C. Baquero, D. Liu, D. He, and J. Liu. A Markov random walk under constraint for discovering overlapping communities in complex networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011, 2011.
19. A. Lancichinetti, S. Fortunato, and J. Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11, 2009.
20. A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 78(4), 2008.
21. J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World Wide Web*. ACM, 2010.
22. J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1), 2007.
23. D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, 2003.
24. P. Massa and P. Avesani. Trust metrics in recommender systems. In *Computing with Social Trust*. Springer London, 2009.
25. M. Newman and M.Girvan. Finding and evaluating community structure in networks. *PHYS.REV.E*, 69:026113, 2004.
26. V. Nicosia, G. Mangioni.V.Carchiolo, and M. Malgeri. Extending the definition of modularity to directed graphs with overlapping communities. *Journal of statistical Mechanics: Theory And Experiment*, 2009.
27. P. Pons and M. Latapy. Computing communities in large networks using random walks. In *ISCIS*, 2005.

28. A. Prat-Pérez, D. Dominguez-Sal, J. M. Brunat, and J.-L. Larriba-Pey. Shaping communities out of triangles. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12. ACM, 2012.
29. M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences of the United States of America*, 105, 2008.
30. S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.
31. SNAP. http://snap.stanford.edu/data.
32. TrustLet. http://www.trustlet.org/.
33. S. M. van Dongen. *Graph clustering by flow simulation.* PhD thesis, University of Utrecht, 2000.
34. J. Xie and B. K. Szymanski. Towards linear time overlapping community detection in social networks. PAKDD'12. Springer-Verlag, 2012.
35. B. Yan and S. Gregory. Detecting communities in networks by merging cliques. *CoRR*, 2012.
36. J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12. ACM, 2012.
37. L. Yen, L. Vanvyve, D. Wouters, F. Fouss, F. Verleysen, and M. Saerens. Clustering using a random-walk based distance measure. In *Proceedings of ESANN'2005*, 2005.
38. S. Zhang, R. S. Wang, and X. S. Zhang. Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A: Statistical Mechanics and its Applications*, 374(1):483–490, 2007.