Community detection in very large dense network with parallel strategy

Zhan Bu
College of Computer Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing, China
buzhan@nuaa.edu.cn

Jiandong Wang
College of Computer Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing, China
aics@nuaa.edu.cn

Zhengyou Xia
College of Computer Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing, China
zhengyou_xia@nuaa.edu.cn

Chengcui Zhang
College of Computer and Information Sciences
The University of Alabama at Birmingham
Birmingham, USA
zhang@cis.uab.edu

***Abstract*— Discovering the latent communities is a useful way to better understand the properties of a network. However, the typical size of virtual spaces is now counted in millions, if not billions, of nodes and edges, most existing algorithms are incapable to analyze such large scale dense networks. In this paper, a fast parallel modularity optimization algorithm that performs the analogous greedy optimization as CNM and FUC is used to conduct community discovering. By using the parallel manner and sophisticated data structures, its running time is essentially fast. In the experimental work, we evaluate our method using real datasets and compare our approach with several previous methods; the results show that our method is more effective in find potential online communities.**

***Keywords- dense network, community detection, parallel, modularity* optimization.**

I. INTRODUCTION

A large number of researches have been devoted to the task of defining and identifying communities in virtual spaces. Most previous papers [1-4] on the subject of community detection in large networks noted that it is a matter of common experience that communities exist in such networks. These papers then argued that, although there is no agreed-upon definition for a community, a community should be thought of as a set of nodes that has more and/or better-connected edges between its members than between its members and the remainder of the network. These papers then apply a range of algorithmic techniques and intuitions to extract subsets of nodes and then interpret these subsets as meaningful communities. These algorithms focus on optimizing an energy-based cost function that is always defined with fixed parameters over possible community assignments of nodes. A notable work proposed by Newman and Girvan [1] introduced modularity as a posterior measure of network structure. This metric has been influential in the community-detection literature and has found success in many applications [3-4].

However, the typical size of virtual spaces is now counted in millions, if not billions, of nodes and edges. Most existing algorithms [1, 5-10] are incapable to analyze large-scale dense networks. In our previous work [3-4], the fastest approximation algorithm for optimizing modularity on large networks (CNM algorithm [1]) was applied to analyze various subsets of a comment relationship network obtained from a bulletin board system (BBS). CNM consists in recurrently merging communities that optimize the production of modularity. It uses a clever data structure to store and retrieve information required to update matrix Q. As the community merge process is always serial, GNM runs well only for a mid-scale subset of the network, it is incapable to analyze larger networks.

In this paper, we propose a fast parallel modularity optimization algorithm that performs the analogous greedy optimization as CNM [1] and FUC [6] to conduct community discovering. In the experimental work, we evaluate our method using real datasets and compare our approach with several previous methods; the results show that our method is more effective in find potential online communities.

II. PREVIOUS WORK

A common formulation of the problem of community detection is to find a partitioning $C = \{C_1, C_2, ...C_\kappa\}$ of disjoint subsets of vertices of the graph $G =< V, E >$ representing the network, in a meaningful manner. Several algorithms have therefore been proposed to find reasonably good partitions in a reasonably fast way. It is common to differentiate by their angles, which create the notions of division-based, agglomeration-based, optimization-based, and label-based algorithms:

a)Divisive algorithms detect inter-community links and remove them from the network. For example, GN algorithm [8] uses edge betweenness as a metric to identify the boundaries of communities. Though it has been applied very successfully to small-scale networks, it makes heavy demands on computational resources, running in $O(n^3)$ time on a sparse network with *n* nodes. The cubic complexity algorithm may not be scalable enough for the size of Online Social Networks. Some other works with this notion can be referenced, such as [9-10].

b) Agglomerative algorithms merge similar nodes/communities recursively. Pons P and Latapy M [5] propose a measure of similarities between vertices based on random walks, which can be used in an agglomerative

algorithm to compute efficiently the community structure of a network. The computational complexity of the Pons-Latapy algorithm is $O(n^2 \log n)$ and space $O(n^2)$ in most real-world cases.

c)Optimization methods are based on the maximization of an objective function. The most famous optimization method is in literature [1], which begins with nodes in n different communities and group together communities which has the greatest contribution to the modularity measure Q. In effect, [1] reduce the time complexity of the algorithm to $O(md \log n)$, where m is the number of edges and d is the depth of the dendrogram obtained. FUC [6] is another heuristic method that finds partitions of a given network by maximizing the modularity measure. It iteratively repeated two phases until to no increase of modularity is possible: one where modularity is optimized by allowing only local changes of communities; one where the communities found are aggregated in order to build a new network of communities. Its complexity is linear on typical and sparse data.

d) LPA (Label Propagation Algorithm) [7] uses only the network structure as its guide, is optimized for large-scale networks, does not follows any a priori defined objective function and does not require any prior information about the communities. In addition, this technique does not need to define in advance the number of communities present into the network or their size. Labels represent unique identifiers, assigned to each vertex of the network. Its functioning is reported as described in [7]. It also takes a near-linear time for the algorithm to run to its completion.

The quality of the partitions resulting from above methods is often measured by the so-called modularity, which the fraction of edges that fall within communities, minus the expected value of the same quantity if edges fall at random without regard for the community structure. One form is given by Newman, which is defined as [1]

$$Q = \sum_i (e_{ii} - a_i^2) = TrE - \| E^2 \| \qquad (1)$$

Where E is a $N \times N$ symmetric matrix whose element e_{ij} is the fraction of all edges in the network that link vertices in community C_i to vertices in community C_j. The trace of this matrix $TrE = \sum_i e_{ii}$ is the fraction of edges in the network that connect vertices in the same community, while the row (or column) sums $a_i = \sum_j e_{ij}$ give the fraction of edges that connect to vertices in community C_i. If the network is such that the probability to have an edge between two sites is the same regardless of their eventual belonging to the same community, one would have $e_{ij} = a_i a_j$.

III. OUR ALGORITHM

The main idea behind our parallel clustering algorithm is following. Assume that we start with a network of n nodes. First, we assign a different community to each node of the network and initialize every community/node with unique labels. Suppose that a community C_i has neighbors $C_1^i, C_2^i, ..., C_{k_i}^i$, we define the "local area" of community C_i as $Local_{C_i} = \{C_i, C_1^i, C_2^i, ..., C_{k_i}^i\}$.Then, we evaluate the gain of modularity that would take place by merging any community pair in $Local_{C_i}$. The community pair with the maximum gain of modularity will be joined as one community (the label of one community will be replaced by the other), but only if this gain is positive. If there is no positive gain, all the community pairs in the "local area" of community C_i remain unchanged. As every merging will result the gain of modularity changing in the "local areas" of the two corresponding communities. We make use of a balanced binary tree as in [1] to keep track the maximum gain of modularity for every community. A similar work proposed by Blondel [6] indicates that the ordering of the nodes does not have a significant influence on the modularity that is obtained, while it may affect the computation time. Therefore, in this paper, we parallel apply this process for all communities and the first pass is then complete. One should also note that by taking parallel processing, the gain of modularity updating may exist access conflict. For example, for community C_i, the maximum gain of modularity in its "local area" are contributed by communities C_i and C_j. While for community C_j, the maximum gain of modularity may be contributed by community pair (C_j, C_k). We cannot merge community pairs (C_i, C_j) and (C_j, C_k) simultaneously. To address this problem, we use a mark array, which stores the current state (busy or free) of every community. To merge a given community pair C_i and C_j, we should first check their current labels and states. Only if their labels are different and all the communities in their "local areas" are free, we can take the next process. When this community pair acquires the authorization, we should then mark all the communities in the "local areas" of C_i and C_j's as busy. By the end of merging, the corresponding communities will be marked as free. We perform this process interactively until there are no more changes and a maximum of modularity is attained (See karate graph in Fig. (2). By construction, the number of meta-communities decreases at each pass, therefore most of the computing time is used in the first passes.

As described above, we start off with each vertex being the sole member of a community of one. The four data structures used in our approach are described as follow:

A balanced binary tree T_i for each community C_i, which stores the gain of modularity ΔQ_{jk} for each community pair in the "local areas" of C_i. So that elements can be found or inserted in

$O(\log(k_i + c_i k_i(k_i-1)/2)) \approx O(\log k_i)$ time. Where c_i' is the clustering coefficient of community C_i, which is defined as $c_i' = \frac{2e_i}{k_i(k_i-1)} = \frac{\sum_{j,k} a_{ij}a_{jk}a_{ki}}{k_i(k_i-1)}$, where $a_{jm}=1$ for two neighbors C_j and C_k of community C_i.

An ordinary vector array with elements a_i.

An label vector array with elements l_i.

An state vector array with elements s_i.

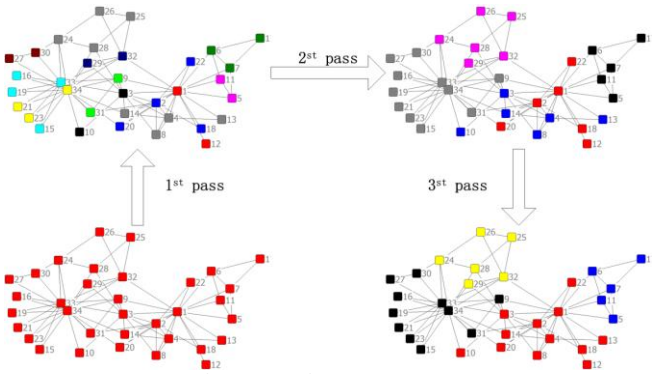

Fig.1 Visualization of the steps of our algorithm on karate network [11]. Each pass is made of parallel merge the community pair (C_i, C_j) with the maximum gain of modularity in the "local area" of every community C_i

Thus, we initially set

$$\Delta Q_{jk} = 1/2m - k_j k_k/(2m)^2,$$

$$if\ C_j, C_k \text{ is int the "local areas" of } C_i \quad (2)$$

$$a_i = k_i/2m \quad (3)$$

$$l_i = i \quad (4)$$

$$s_i = free \quad (5)$$

for each community C_i.

The modularity updating is the as in [1]. If we join communities C_x and C_y, the degrees (the numbers of neighboring communities) of C_x and C_y can be denoted as k_x and k_y. If $k_x > k_y$, we label the combined community x, and y otherwise. The updating rules are as follows. If community C_k is connected to both C_x and C_y, then

$$\Delta Q'_{xk} = \Delta Q'_{kx} = \Delta Q_{xk} + \Delta Q_{yk} \quad (6a)$$

If C_k is connected to C_x but not to C_y, then

$$\Delta Q'_{xk} = \Delta Q'_{kx} = \Delta Q_{xk} - 2a_y a_k \quad (6b)$$

If C_k is connected to C_y but not to C_x, then

$$\Delta Q'_{xk} = \Delta Q'_{kx} = \Delta Q_{yk} - 2a_x a_k \quad (6c)$$

We take the similar method as [1] to analyze the computational complexity of FPCDA. Take community C_i as an example. First, we need to select the community pair (C_x, C_y) with the maximum gain of modularity from C_i's balanced binary tree T_i, it will take $O(1)$ time. Then, we need to join the two communities as one, at the same time, we need update corresponding balanced binary trees. We will take a heuristic method to update the balanced binary tree T_x: 1) For every element in T_x, we give them a same decrement ($-2a_y a_k$) without changing the tree structure to complete Eq. (6b) first. And it will take $O(k_x)$ time. 2) To implement Eq. (6a), we insert the elements of T_y into T_x, summing them wherever an community connects to both C_x and C_y. It is worth noting that we should also add a same increment ($2a_y a_k$) to those elements, as we additionally subtract it in Step 1). Each of this $count(N_x \cap N_y)$ insertions takes $O(\log k_x)$ time. 3) We then update the other elements of T_x, of which there are at most $count(N_y - N_x \cap N_y)$, according to Eq. (6c). The total operate time for T_x is $O(k_x + k_y \log k_x)$. In the balanced binary trees T_k, we will update a single element, taking $O(\log k_k)$ times, and there are at most $k_x + k_y$ trees for us to update. Finally, the updates $a'_x = a_x + a_y$ and $l'_x = l_x + l_y$ is trivial and can be done in constant time.

By construction, the modularity updating (Steps 30-33 in FPCDA) can be also applied in parallel way. After T_k has been updated, we free the corresponding community C_k, then other community may have the chance take the merge operating. For the pth pass, FPCDA takes $O(k_p^{\max}(< k_p > + \log k_p^{\max}))$ time, where $k_p^{\max}$ and $< k_p >$ are the maximum degree and the average degree of the network in the pth pass. The total time of FPCDA is $O(\sum_{p=1}^{d} k_p^{\max}(< k_p > + \log k_p^{\max}))$, where d is the number of passes. From our experiments, we found that the main computational time comes from the first pass and 95% of the nodes or more are classified correctly by the end of the sixth pass. This is agreed with the well-known "six degree of separation" theory. However, the mathematical convergence is hard to prove. Therefore, FPCDA has a running time of $O(k^{\max}(k^{\max} + < k > \log k^{\max}))$.

IV. COMPARISON AND EXPERIMENT

In order to verity the validity of FPCDA, we have applied it on a number of test-case networks and compared it with four other community detection algorithms. In all cases, one can observe that the rapidity and the large values of the modularity that are obtained. FPCDA outperforms nearly all the other methods to which it compared, expect for Gplus. That is because the maximum degree of Gplus network is very high, FPCDA may degenerate to serial one. We also have applied FPCDA on the four networks derived from Tianya data [3]. As shown in the last four rows of Table 1, even for those very dense networks (SVN and IN), FPCDA

performs very well (17.7s and 80.3s, respectively). In the above examples, the number of passes is always smaller than 6.

TABLE I. THE PERFORMANCES OF THE ALGORITHM OF PL [5], CNM [1], FUC [6], LPA [7], AND OUR ALGORITHM FOR COMMUNITY DETECTION IN NETWORKS OF VARIOUS SIZES. MOST OF THE DATA WE USED ARE DOWNLOADED FROM STANFORD LARGE NETWORK DATASET COLLECTION (HTTP://SNAP.STANFORD.EDU/DATA/). FOR EACH METHOD/NETWORK, THE TABLE DISPLAYS THE MODULARITY THAT IS ACHIEVED AND THE RUNNING TIME. EMPTY CELLS CORRESPOND TO A RUNNING TIME OVER 24H. OUR METHOD CLEARLY PERFORMS BETTER IN TERMS OF RUNNING TIME AND MODULARITY.

	nodes/edges	$<k>/k^{max}$	PL	CNM	FUC	LPA	FPMQA
Karate [11]	34/77	4.5/16	**0.42/0s**	0.38/0s	**0.42/0s**	0.38/0s	**0.42/0s**
Facebook [12]	4k/88k	43.7/518	0.76/27.9s	0.72/22s	0.82/0s	0.75/0s	**0.84/0s**
Twitter [12]	81k/1.8M	43.5/875	0.70/733s	0.71/818s	**0.73**/2.3s	0.69/3.1s	0.72/**1.3s**
Gplus [12]	108k/13.7M	254.1/5126	0.66/3657s	0.62/3321s	0.69/69.5s	0.65/**23.8s**	**0.71**/45.7s
LiveJournal[13]	4M/34.9M	17.3/1087	_/_	_/_	0.14/152.4s	0.12/60.3s	**0.15/2.0s**
Orkut [13]	3M/117M	76.2/3295	_/_	_/_	0.19/437s	0.18/203s	**0.20/18.9s**
Friendster [13]	117M/2.6B	44.4/2768	_/_	_/_	_/_	0.22/165mn	**0.24/133s**
SVN [4]	155k/15.3M	198.2/3209	0.33/8275s	0.35/7833s	**0.36**/183.2s	0.34/46.3s	**0.36/32.7s**
IN [4]	319k/40.4M	253.8/5918	_/_	**0.36**/6.5h	0.35/287.3s	0.34/109.6s	0.35/**89.3s**
Und. DN [2]	324k/3.0M	18.5/6505	_/_	0.49/1731s	**0.50**/163.2s	**0.50/10.2s**	**0.50**/72.8s
SN (3)	163k/678k	8.3/3504	0.58/7535s	0.60/350s	**0.62**/49.4s	0.58/**5.2s**	**0.62**/21.2s

V. CONCLUSIONS

In this paper, we propose a fast parallel modularity optimization algorithm for inferring community structure from network topology which performs the analogous greedy optimization as CNM [1] and FUC [6]. It combines the advantages of CNM [1] and FUC's [6] and has several highlights. CNM finds the pair of communities with the "global" maximum ΔQ, however, finding this "global" maximum ΔQ is time consuming. Our method select the "local" maximum ΔQ as FUC does, but the definition of the "local area" of every community is more reasonable. FUC only considers the community pair between C_i and its neighboring communities. That may bring the final modularity declination in some cases, as stated in Section 4.3. What's more, our algorithm takes the parallel strategy, it has a running time of $O(k^{\max}(k^{\max} + < k > \log k^{\max}))$. It can be applied to the community detection task in a large scale dense network, if the maximum degree of the network is not very high.

ACKNOWLEDGMENT

This work was supported by Funding of Jiangsu Innovation Program for Graduate Education (Project No: CXZZ12_0162), the Fundamental Research Funds for the Central Universities and Shanghai Intelligent Information Processing Laboratory (Subject No: IIPL-2011-007) and supported by the NUAA Fundamental Research Funds (Subject No.NS2013090).

REFERENCES

[1] A. Clauset, M. E. J. Newman, and C. Moore, "Finding and evaluating community structure in networks", PHYSICAL REVIEW E. 69, 026113, 2004.

[2] V. Gomez, A. Kaltenbrunner, and V. Lopez, "Satistical analysis of the social network and discussion threads in Slashdot", Proceedings of the 17th international conference on World Wide Web, pp. 645-654, doi: 10.1145/1367497.1367585.

[3] Z. Xia, Z. Bu, "Community detection based on a semantic network", Knowledge Based Systems, vol. (26) pp. 30–39, 2012.

[4] Z. Bu, Z. Xia and J. Wang, "A sock puppet detection algorithm on virtual spaces", Knowledge Based Systems, vol. (37), pp. 366-377, 2013.

[5] P. Pons and M. Latapy, "Computing communities in large networks using random walks", Journal of Graph Algorithms and Applications, vol. 10(2), pp. 191-218, 2006.

[6] V. D. Blondel, J. L. Guillaume, R. Lambiotte and E. Lefebvre, "Fast unfolding of communities in large networks", Journal of Statistical Mechanics: Theory and Experiment ,Volume, 2008.

[7] U. N. Raghavan, R. Albert and S. Kumar, "Near linear time algorithm to detect community structures in large-scale networks", PHYSICAL REVIEW E. 76, 036106, 2007.

[8] M. Girvan and M.E.J. Newman, "Community structure in social and biological networks", PNAS, vol. 99(12), pp. 7821-7826, 2002.

[9] M.E.J. Newman, "Detecting community structure in networks", THE EUROPEAN PHYSICAL JOURNAL B - CONDENSED MATTER AND COMPLEX SYSTEMS, vol. 38(2), pp. 321-330, 2004.

[10] L. Danon, J. Duch, A. Diaz-Guilera and A. Arenas, "Comparing community structure identification", Journal of Statistical Mechanics: Theory and Experiment, Volume, 2005.

[11] Zachary, W. W. (1977) J. Anthropol. Res. 33, 452–473.

[12] J. McAuley and J. Leskovec, "Learning to Discover Social Circles in Ego Networks", Neural Information Processing Systems, 2012.

[13] J. Yang and J. Leskovec. "Defining and Evaluating Network Communities based on Ground-truth". ICDM, 2012.