EXTRACTING SUBPOPULATIONS FROM LARGE SOCIAL NETWORKS

Bin Zhang
Machine Learning Department
Carnegie Mellon University
Pittsburgh

Data Analysis Project

Committee:
Dr. William Cohen
Dr. David Krackhardt (Chair)
Dr. Ramayya Krishnan

February 21, 2011

Abstract

Until recently, collecting network data of any size was a challenge and limited much of the research to analyzing relatively small networks (less than 1000 nodes). Many of our analytic tools worked fine on networks of this size. Now, however, with the help of new information technologies, we find ourselves having access to very large data sets, perhaps on the order of hundreds of thousands of nodes to even billions of nodes. This large size has outstripped our ability to perform even rudimentary analysis on the networks as a whole. One solution to this problem is to extract connected subgraphs from this large network and analyze their properties as stand-alone subpopulations. We propose a method for extracting such a subpopulation from a large population in such a manner that two desirable properties are maintained: 1) that it be *effective*, resulting in subpopulations with more ties within them than to nodes outside each subpopulation; and 2) that it be fast, so that it scales well to large networks. We develop a method for such extractions, called the "Transitive Clustering and Pruning" (T-CLAP) algorithm. We compare the speed and effectiveness of this algorithm to two other popularly community detection algorithms in the literature – Newman's and Clauset's community detection algorithms. We find that T-CLAP and Newman's algorithm both are effective, but that Newman's algorithm is orders of magnitudes slower than T-CLAP. We find that T-CLAP and Clauset's algorithm are both very fast and scale well, but that T-CLAP is superior to Clauset's algorithm in terms of returning effective subpopulations that are useful to study.

Contents

1 Introduction

The study of network analysis currently faces a unique challenge of analyzing large, even huge, networks. Examples include networks of all cell phone calls through a call center and scientific coauthorship and citation networks for all of science (across multiple disciplines). There has been a large amount of social network research of relatively small networks. One reason has been an historical interest in smaller systems. Another is that collecting network data was difficult to collect at that time which reinforced attention to smaller systems. However, contemporary data collection and retrieval technologies, together with the pervasiveness of online social networks give us access to networks with up to billions of nodes. However, much of current social network analysis (SNA) software cannot handle large network data. Some methods (*e.g.* SIENA, pstar, and QAP) are quite computationally burdensome so that analysis for networks whose size is 10^4 is intractable. Therefore, one thing we may want to do, instead of studying 'the' whole network, is to extract subpopulations from the network and study interesting aspects of them in a comparative fashion[1]. Often, we need networks with a relatively small node size *e.g.*, less then 1000, that are internally well connected, but at the same time do not have many ties to the external network. In large measure, we do not want 'boundary leakage' (defined below) with the external network contaminating the structure of the extracted networks. Some literatures have similar goals to ours, for example community detection. However they want to partition the graph, which is different from our goal which is extracting subpopulations that have reasonable properties for further study. We present a method that can extract subpopulations from large scale networks quickly and with minimal boundary leakage. We also compare our method with two popular methods having similar objectives in terms speed and effectiveness.

The rest of the paper is organized as follows. We discuss the literature on subpopulation extraction, for example snowball sampling, Modularity Spectral Optimization, and local modularity maximization in Section 2; Our subpopulation extraction method, one that can identify local communities quickly and with low computing cost, is presented in Section 3. In Section 4 we present the results of empirical comparisons between our method, spectral clustering, and local modularity maximization. Conclusions and suggestions for future work complete the paper in Section 5.

2 Literature Review

Increasingly, social network researchers face a challenging problem before doing any analysis: making sure their work is tractable with respect to their data. Given that we are in an era of advanced data collection and management, the potential size of our data sets can be quite large. We often have networks with node sizes in the millions with an edge size in the tens of millions. Networks with billions of nodes are possible. Since it is not feasible to apply most existing analysis methods to handle networks of this size, it is useful to extract coherent subpopulations from the full data. Three current methods are: snowball sampling, spectral modularity clustering and greedy maximization of local modularity.

[1]Programs such as pajek that can handle very large networks, for example, (Batagelj and Mrvar, 1998) often break large networks into smaller parts meriting further attention.

2.1 Snowball Sampling

Often, when an original network is large, it is necessary to construct or identify subpopulations and use the subpopulations for our analyses. We know that random sampling of social network nodes does not preserve the network's structure at the local level. A more legitimate and efficient method to build one (or more) subpopulation(s) in this situation is snowball sampling. This method was introduced by Coleman (1958) and Goodman (1961). Individuals in the sample are identified through a chain-referral process and data are collected from them. The sampling procedure starts with predefining the number of steps of sampling, denoted by s, then randomly drawing some nodes from the population. For each node i, we get k nodes that it connects to, with k a pre-specified number of nodes. Sampling stops after s steps. Since s and k are all defined before the sampling, it is also called s-stage k-name snowball sampling. It is possible that nodes linked from one node will be linked from other nodes again. This method has been extended by Salganik and Heckathorn (2004). In their method, an individual sample is formed by randomly selecting a user from the network and returning the connected component containing this user, repeating this on the remaining users until some maximum number of users is attained. This method is desirable for social network data sampling since it allows researchers to have larger sample size than other methods given available resources (Semaan et al., 2002). Snowball sampling is also referred to as chain-referral sampling, link-tracing sampling, and random-walk sampling. Samples can be used to make estimates about the network connecting the population. Using information about networks constructed from snowball samples, we can "derive the population proportion in different groups" (Salganik and Heckathorn, 2004).

However, snowball sampling has shortcomings. For example any bias in the seed selection would lead to a biased sample. The estimates drawn from snowball samples are biased and cannot be used to make inferences about the whole population. Even so, it may be still suitable in our subpopulation extraction task because we are more interested in the impact of network structure on actor behaviors. Snijders (1992) points out that snowball sampling is more appropriate for inference about the structure of the network. Since our research concentrates on the network effect of each node, snowball sampling's breadth-first search principle could be part of a subpopulation extraction algorithm.

2.2 Modularity Spectral Optimization (Newman)

There are subpopulation extraction methods in the cluster detection literature. This is a field attracting researchers in physics, mathematics, and computer science. It has been found that many networks are inhomogeneous, consisting of distinct groups "with dense connections within groups and only sparser connections between them" (Newman, 2004). Such groups are called "communities". Researches show that communities at the local level can be quite different from each other, and even different from the global network (Newman, 2006b). So analyses on local communities can give us more properties of, and information about, the network. Here, we would like to consider modularity spectral optimization as an alternative method. Modularity Q is the best known quality measure for community detection has been shown to be effective. It features the difference between the actual density of edges in a subgraph and the expected density of edges in that subgraph. If such a difference is large, meaning the number of edges within such subgraph is more dense then expected, and the subgraph is deemed cohesive. Modularity

spectral optimization is a community detection method using modularity as an evaluation criteria and optimized by using eigenvalues (spectrum) and eigenvectors of a modularity matrix. The definition of the modularity matrix is first given in Girvan and Newman (2002). As shown below: modularity, Q, is difference between the actual number of edges in communities and the expected number of edges. If it is large, meaning the edges are more dense then expected, the communities is cohesive. Note that i and j must belong to the same group. If the difference between actual and expected edge between two nodes that belong to same group is large then Q is large. The definition of Q is described as below. Let m be the total edges of the graph, A_{ij} be the actual number of edges between nodes i and j, P_{ij} be the expected number of edges between i and j; g_i be the group's number that node i belongs to; g_j be j's group number; and $\delta(g_i, g_j)$ indicates whether i and j belong to the same group, where 1 stands for the same group and 0 otherwise.

$$Q = \frac{1}{2m} \sum_{i=1}^{n} \sum_{j=1}^{n} [A_{ij} - P_{ij}] \delta(g_i, g_j)$$

Newman further developed the modularity spectral optimization method. The detailed implementation of the algorithm is described in Newman (2006a). In such method, Q is transformed as the product of modularity matrix $\mathbf{B}$ and index vector $\mathbf{s}$ shown as

$$Q = \frac{1}{4m} \mathbf{s}^\top \mathbf{B} \mathbf{s} \tag{1}$$

where $B_{ij} = A_{ij} - P_{ij}$; $s_j = +1$ if node j belongs to the first block; $s_j = -1$ if node j belongs to the second block. Since the right hand side of Equation (1) is a standard matrix product, $\mathbf{s}$ can decomposed as

$$\mathbf{s} = \sum_i a_i \mathbf{u}_i, \text{ where } a_i = \mathbf{u}_i^\top \mathbf{s}$$

where u_i are eigenvectors of modularity matrix $\mathbf{B}$.

We then have

$$\begin{aligned} Q &= \frac{1}{4m} \sum_i a_i \mathbf{u}_i^\top \mathbf{B} \sum_j a_j \mathbf{u}_j \\ &= \frac{1}{4m} \sum_i a_i^2 \beta_i \\ &= \frac{1}{4m} \sum_i (\mathbf{u}_i^\top \mathbf{s})^2 \beta_i \end{aligned} \tag{2}$$

where β_i is the correspondent eigenvalue of $\mathbf{u}_i$.

The modularity bipartition problem is now approximated by the bisection problem of eigenvector corresponding to the leading eigenvalue. In this case, all the nodes that have positive component value in eigenvector will be in the first block, and the other nodes will be in the second block. So the modularity Q can be represented as the product of it eigenvalue and eigenvector thus the computation can be reduced significantly. However, even with spectral optimization, the Newman algorithm is still intractable when the population is large. Thus we have to look at other methods.

2.3 Greedy Maximization of Local Modularity (Clauset)

The greedy maximization of the local modularity algorithm designed by Clauset is considered to be one of the fastest community extraction algorithms. We refer this algorithm as the Clauset algorithm hereafter. Clauset (2005) defines the following node sets: $\mathcal{C}$ is the known portion of the graph which has been included in the subpopulation; $\mathcal{U}$ is the set of nodes in the external network adjacent to $\mathcal{C}$; and $\mathcal{B}$ is the subset of $\mathcal{C}$ having at least one neighbor in $\mathcal{U}$. A sharp boundary's definition is given by: few connections from $\mathcal{B}$ to $\mathcal{U}$, while having more connections from $\mathcal{B}$ to $\mathcal{C}$. Starting from a random node in the population, the Clauset algorithm moves nodes maximizing local modularity from $\mathcal{U}$ to $\mathcal{B}$, and eventually to $\mathcal{C}$. The manner of adding a new node is similar to that of an internet crawler. The local modularity is defined as R.

$$R = \frac{\sum_{ij} B_{ij}\delta(i,j)}{\sum_{ij} B_{ij}} = \frac{H}{T}$$

where

$$B_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are connected, } i \text{ in } \mathcal{B}, \text{ or } j \text{ in } \mathcal{B} \text{ or both} \\ 0 & \text{otherwise} \end{cases}$$

$$\delta(i,j) = \begin{cases} 1 & \text{if one node is in } \mathcal{C} \text{ and the other is in } \mathcal{B} \\ 0 & \text{otherwise} \end{cases}$$

H is the number of edges between $\mathcal{B}$ and $\mathcal{C}$, and T is the number of edges in $\mathcal{B}$, between $\mathcal{B}$ and $\mathcal{C}$, and between $\mathcal{B}$ and $\mathcal{U}$.

In each step, the algorithm moves a node j maximizing the change in local modularity ΔR from $\mathcal{U}$ to $\mathcal{C}$, until a predefined threshold n_k is reached. The calculation of ΔR_j can be simplified as getting the change in the number of edges in the three node sets caused by moving j.

$$\begin{aligned} \Delta R_j &= R_{+j} - R \\ &= \frac{I_{+j}}{T - z + y} - R \\ &= \frac{x - Ry - z(1-R)}{T - z + y} \end{aligned}$$

where x is the number of edges to nodes in $\mathcal{B}$, y is the number of edges to new nodes in $\mathcal{U}$. The Clauset algorithm in general is much faster than the Newman algorithm. The complexities of these two algorithms are presented in the section below.

2.4 Algorithm complexities

The time complexity of graph partitioning method using modularity as quality fucntion is not fast if we have huge network. For example, the complexity of spectral clustering method to bisect a graph is $O(n^3)$, where n is the number of nodes in the network. The spectral optimization method can reduce

complexity to $O(nm\ log(n))$, or $O(n^2\ log(n))$ in a sparse matrix, if we recursively bipartition the network until each node is a community. Often we do not need to partition graph until the single node level, so the lowest possible complexity of such algorithm is $O(nm)$, where n is the total number of nodes in the population and m is the total number of edges in the population. However, if the number of nodes is at the level of, or beyond one million, such a method is still not tractable. Summarizing the current status of community detection method complexity, Newman writes:

> "I don't think the spectral algorithm will work for such a large network [N=1 million]. Remember that the algorithm is $O(mn)$, where m is edges and n is vertices, so you're talking about 10^{15} operations to perform the eigenvector calculation, which is not feasible with current computers.
>
> Basically, if you're working with networks that large, you are limited to $O(n)$ or $O(nlogn)$ algorithms, of which there are only a few, none of which work very well. There was a new multi-scale method that came out in the last year that might be worth looking at, and there's an improved version of the greedy algorithm of Clauset et al. that can do a good job in some circumstances. But overall the situation is not very promising for extremely large networks at present."[2]

Such concerns highlights the challenge of this topic; there is no method that can really solve this problem with a large-scale network. Some well-accepted methods have to make a trade-off between speed and stability. As Newman argues about the performance of his spectral clustering method.

> "I should point out that this code should not be used for gauging the speed of the algorithm. It uses dense-matrix methods to do the eigenvector calculation, which are very slow. A much faster implementation is possible using sparse matrix methods. I have such an implementation, but it's not very stable. For merely testing the efficacy of the algorithm on small networks this implementation is better."[3]

Clauset (2005) proposed a community detection method that achieves which improves the local modularity maximization by using a greedy algorithm. This method's complexity is $O(n^2d)$, where n is the number of nodes traversed and d is the mean degree of node. But subpopulations returned by using this method do not have good quality.

Since none of snowball sampling, modularity spectral optimization or local modularity optimization can extract subpopulations with the ideal balance of speed and quality, we propose a different method – Transitive Clustering and Pruning (T-CLAP) Algorithm – and we return to time complexity after presenting this algorithm.

[2] Personal communication between Newman and the authors

[3] *Ibid.*

3 Transitive Clustering and Pruning (T-CLAP) Algorithm

We present a subpopulation extraction method that can identify dense and isolated local communities *without* processing the whole network. Our goal is to find subpopulations having the following three characteristics. First, they have high internal density.) Second, these subpopulations are reasonably dense but not complete. (We need to find dense subpopulations, but not so dense that we lose variation of nodal degree. Third, they have relatively few ties from within them to the rest of the network. The result is that each subpopulation is relatively isolated and integral in itself. The criteria we use to evaluate the quality of the returned subpopulations is the *I-E* ratio, which is the ratio between the difference of total internal edges and external edges of a block and the summation of these two sets of edges. The formal terms we use are listed in Table 1.

Table 1: Symbols used to describe the T-CLAP algorithm

Symbol	**Definition and Description**
v_0	Source node of the initial BFS sampling
$\mathcal{C}$	Core of subpopulation
$\mathcal{B}$	Boundary of subpopulation
$s_1^{\mathcal{C}}$	Predefined stopping rule for the initial Core
$s_f^{\mathcal{C}}$	Predefined stopping rule for the final Core
$n_{\mathcal{C}}$	Node size of core
$n_{\mathcal{B}}$	Node size of boundary
n_θ	Node size of BFS seed set
$n_{\mathcal{S}}$	Node size of subpopulation extracted
n_k	Number of nodes to be explored (the Clauset algorithm)
$k_{\mathcal{S}}^{int}$	Total internal degree of subpopulation extracted (I)
$k_{\mathcal{S}}^{ext}$	Total external degree of subpopulation extracted (E)
I-E	Ratio of $(k_{\mathcal{S}}^{int} - k_{\mathcal{S}}^{ext})/(k_{\mathcal{S}}^{int} + k_{\mathcal{S}}^{ext})$
t	Repeat times of BFS sampling before pruning
r	Cycle of recalculating $k_j^{int} - k_j^{ext}$ for each node j in core during pruning stage
n_b	Node size of the block in the data
k_b^{int}	Total internal degree of a block
$\delta(b)$	Density of a block
n	Node size of the population
m	Edge size of the population

Our method is designed to find subpopulations having the preferred characteristics as above and consists of two steps, clustering and pruning.

3.1 Clustering

Our method starts with a randomly selected node v_0 in the global network $\mathcal{G}$. We define an initially empty set, *core*, which is denoted by $\mathcal{C}$. Throughout. *Core* consists of candidate nodes that a future subpopulation will contain. We add v_0 to $\mathcal{C}$, and use the Breadth First Search[4] (BFS) principle to add all neighbors of v_0, $N(v_0)$, to $\mathcal{C}$. $N(v_0) = u_1, u_2, ..., u_a$, $|N(v_0)| = a$. Starting from first neighbor of v_0, u_1, we get all of its neighbors, denoted by $N(u_1)$, where $N(u_1) = \{u_{a+1}; u_{a+2}, ..., u_{a+b}\}$, $|N(u_1)| = b$. And then get all the neighbors of u_2, $N(u_2) = \{u_{a+b+1}, u_{a+b+2}, ..., u_{a+b+c}\}$, $|N(u_2)| = c$. For each neighbor of v_0, we add all of its neighbors to $\mathcal{C}$. We repeat by including new nodes in the *core* until a stopping rule $s^{\mathcal{C}}$ is met, which is the predefined number of nodes. Assuming, without loss of generality, node u_s, the $s^{\mathcal{C}}$-th node added to $\mathcal{C}$, is d distance away from the source node v_0, we also add all the new nodes having distance d from v_0 to $\mathcal{C}$. The final size of such a provisional *core* is $n_{\mathcal{C}}$, and $n_{\mathcal{C}} \geq n$. The reason for getting $n_{\mathcal{C}}$ nodes instead of $s^{\mathcal{C}}$ is that we want to get a completed tree, and record more complete connection among nodes included in our subpopulations. The BFS sampling is shown in Figure 1.

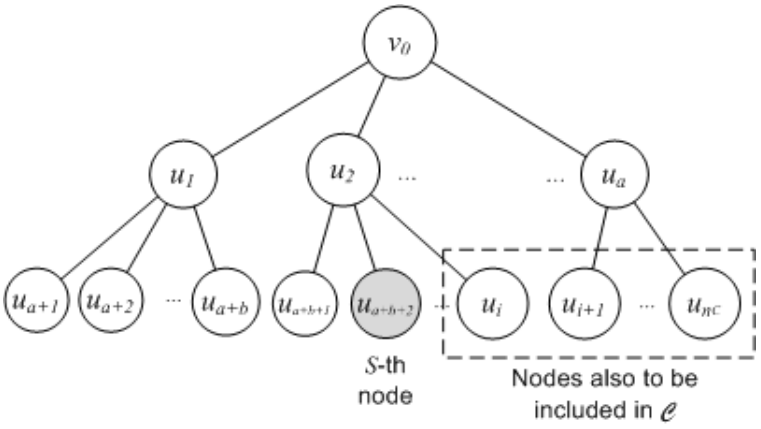


Figure 1: BFS Sampling

We then define the set of nodes in the external network that are neighbors of nodes in $\mathcal{C}$ as a *boundary*, which can be understood as the boundary between our *core* and the external network. Denote the boundary as $\mathcal{B}$, which has a size of $n_{\mathcal{B}}$. We also denote the internal degree of $\mathcal{C}$, the total number of edges in $\mathcal{C}$, as $k_{\mathcal{C}}^{int}$; the external degree of of $\mathcal{C}$, the total number of edges between $\mathcal{C}$ and $\mathcal{B}$, as $k_{\mathcal{C}}^{ext}$. For illustrations of each variable, see Figure 2.

We then construct two adjacency matrices, $\mathbf{A}$ and $\mathbf{B}$. $\mathbf{A}$ is the adjacency matrix for the nodes in $\mathcal{C}$. In

[4]BFS sampling is often referred as snowball sampling.

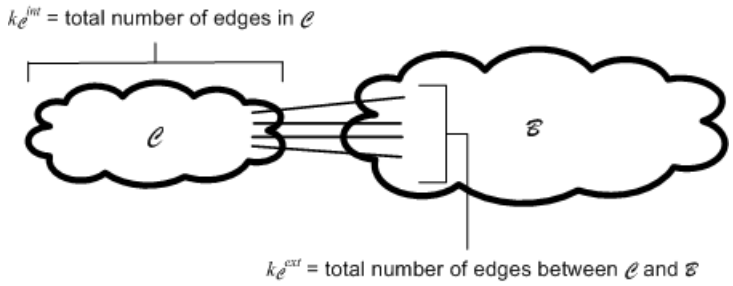


Figure 2: Illustration of subpopulation core $\mathcal{C}$ and its boundary $\mathcal{B}$

A, the element A_{ij} shows whether node i is linked to node j or not.

$$A_{ij} = \begin{cases} 1 & \text{if node } i \text{ connects to node } j, \\ & \text{and both nodes are in } \mathcal{C} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{where } i, j \in [1, n_{\mathcal{C}}]$$

The second matrix **B** is the adjacency matrix for nodes in $\mathcal{C}$ and $\mathcal{B}$:

$$B_{ij} = \begin{cases} 1 & \text{if node } i \text{ connects to node } j, \\ & \text{and either node is in } \mathcal{C} \text{ and the other is in } \mathcal{B} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{where } i \in [1, n_{\mathcal{C}}] \text{ , and } j \in [1, n_{\mathcal{B}}],$$

We then concatenate matrices **A** and **B** into one large matrix [**A B**] (See Figure 3). This joined adjacency matrix records ties for pairs of nodes where at least one of them belongs to $\mathcal{C}$. A value of 1 in columns 1 to $n_{\mathcal{C}}$ indicating two nodes in $\mathcal{C}$ are connected, and 0 otherwise; a 1 in columns $n_{\mathcal{C}} + 1$ to $n_{\mathcal{C}} + n_{\mathcal{B}}$ indicates edges between node in $\mathcal{C}$ and in $\mathcal{B}$, and 0 meaning no such edges.

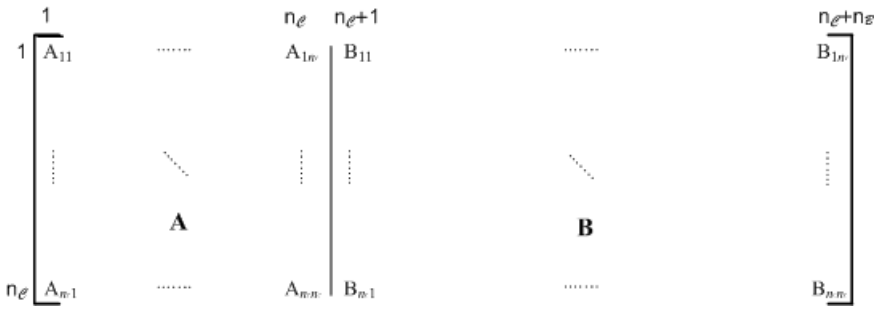


Figure 3: Adjacency matrices **A** and **B**

The product matrix $\mathbf{\Delta} = [\mathbf{A}\ \mathbf{B}] \cdot [\mathbf{A}\ \mathbf{B}]^\top$ gives us the total common connections between nodes in $\mathcal{C}$ (Figure 4). This matrix is of size of $n_\mathcal{C} \times n_\mathcal{C}$. The diagonal element δ_{ii} is the degree of each node i, $i \in [1, n_\mathcal{C}]$, while the off-diagonal element δ_{ij} are the number of common neighbors shared by node i and j. The diagonal of the Hadamard product (element-wise product) $\mathbf{\Xi} = \mathbf{A} \circ \mathbf{\Delta}$ gives us the number of common neighbors between two nodes in $\mathcal{C}$, and they must also be connected. We then find a predefined n_Θ of connected nodes in $\mathbf{\Xi}$ that have the highest degree, and define this set of nodes as a seed set Θ. Note that, because of the nature of our algorithm, $\mathcal{C}$ and $\mathcal{B}$ are different in each clustering round, thus their node sizes $n_\mathcal{C}$ and $n_\mathcal{B}$, respectively, differ also.

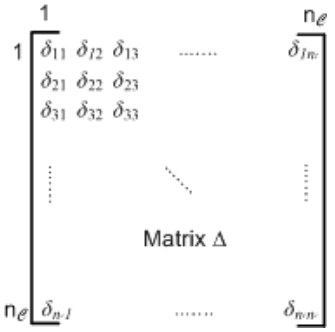


Figure 4: Matrix Δ, product of $[\mathbf{A}\ \mathbf{B}] \cdot [\mathbf{A}\ \mathbf{B}]^\top$

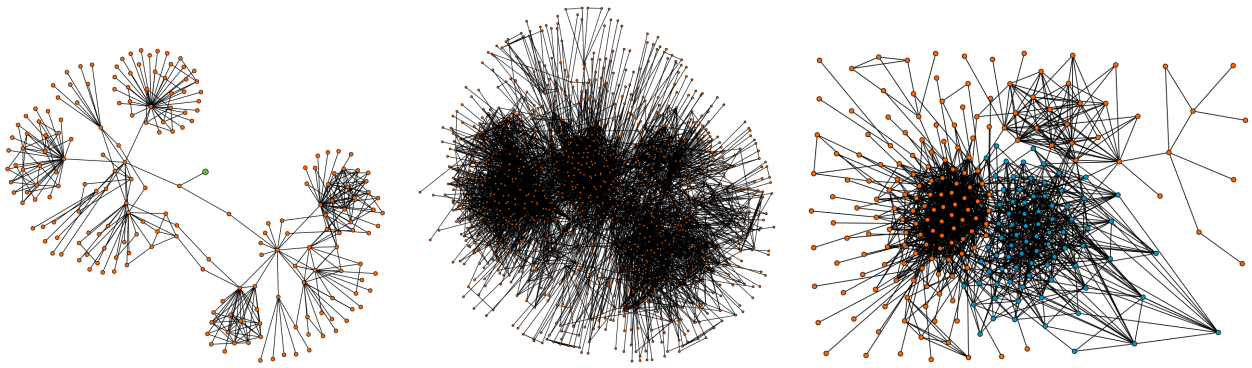
Using nodes in Θ as source nodes, we start a new round of BFS searching, keeping stopping rule $s^\mathcal{C}$ the same as the last round, and including all the new nodes that have the same distance d as the s-th node away from the nodes in $\mathcal{C}$. We get the (new) matrices $\mathbf{A}$ and $\mathbf{B}$ again, and (the new) $\mathbf{\Delta}$ and $\mathbf{\Xi}$ as well. Then we find connected nodes with highest degree in $\mathbf{\Xi}$ to construct Θ. After several rounds of clustering, Θ converges and does not change. Then we construct the final $\mathcal{C}$ and $\mathcal{B}$. (The final stopping rule for BFS sampling can also be adjusted as a parameter, which can be set to be larger than s to include more candidate nodes in a subpopulation, and set to be smaller than s to make the algorithm run faster.) After this clustering stage, we move to the most dense area locally, but will also include nodes having stronger ties to the external network than to internal network $\mathcal{C}$. Define the $\mathcal{C}$ and $\mathcal{B}$ from the final round of clustering as $\mathcal{C}_f$ and $\mathcal{B}_f$ and we move on to pruning stage.

3.2 Pruning

For each node j in $\mathcal{C}_f$, we define the number of nodes in $\mathcal{C}_f$ that j connects to as its internal degree k_j^{int}, and the number of nodes in $\mathcal{B}_f$ that j connects to as its external degree k_j^{ext}. We also define the total number of edges in $\mathcal{C}_f$ as the subpopulation internal degree $k_\mathcal{S}^{int}$, and the total number of edges between $\mathcal{C}_f$ and $\mathcal{B}_f$ as subpopulation external degree $k_\mathcal{S}^{ext}$. Then for all nodes in $\mathcal{C}_f$, we keep pruning the nodes having the smallest $k_j^{int} - k_j^{ext}$, which means pruning the nodes have the lowest value of difference between internal connections than external connections. We also recalculate k_j^{int} and k_j^{ext} for each node j after removing r nodes. We use the I-E ratio, $I\text{-}E = \dfrac{k_\mathcal{S}^{int} - k_\mathcal{S}^{ext}}{k_\mathcal{S}^{int} + k_\mathcal{S}^{ext}}$, as the measure to evaluate the

quality of the subpopulation. Such ratios ranges from -1 to $+1$, where $+1$ occurs when all the edges in the subpopulation are internal, and -1 occurs when all the edges link to nodes in the external network. This measure is a revised version of Krackhardt and Stern (1988)'s *E-I* index. We stop pruning when *I-E* ratio reaches its maximum.

Figure 5 illustrates of our T-CLAP algorithm with three network forms. Figure 5(a) is an initial subpopulation extract before the clustering stage. This subpopulation is not densely connected. Figure 5(b) is the subpopulation after enough rounds of clustering. We get a very dense subpopulation, but also include many nodes that have more edges to external network than internal network, which we call boundary leakage. The subpopulation after pruning is shown in Figure 5(c). The subpopulation is much more dense than the initial one, which means the subpopulation is self-contained.



(a) An initial subpopulation $\mathcal{S}$ at size of 200, before clustering

(b) $\mathcal{S}$ after clustering

(c) $\mathcal{S}$ after pruning

Figure 5: Initial subpopulation and final subpopulation after pruning

Using breath-first search to construct $\mathcal{C}$ has a time complexity $O(n_{\mathcal{C}})$. Constructing the matrix $\mathbf{\Delta}$ has a running time of $O(n_{\mathcal{C}}^2(n_{\mathcal{C}} + n_{\mathcal{B}}))$, or $O(n_{\mathcal{C}}^2)$ in a sparse network; and $\mathbf{\Xi}$ requires $O(n_{\mathcal{C}}^2)$. The complexity of getting a seed set Θ is $O(n_\Theta n_{\mathcal{C}}^2)$. Since the size of Θ is always a constant, the complexity is actually at $O(n_{\mathcal{C}}^2)$. So the total complexity of clustering stage is $O(n_{\mathcal{C}} m_{\mathcal{C}} + t(n_{\mathcal{C}}^2(n_{\mathcal{C}} + n_{\mathcal{B}}) + n_\Theta n_{\mathcal{C}}^2))$, where t is the repeat times of BFS search in clustering stages in order to find a converged seed set Θ. Since it always takes limited times of clustering for Θ to converge, the complexity is $O(n_{\mathcal{C}}^2(n_{\mathcal{C}} + n_{\mathcal{B}}))$. Given a sparse matrix, the complexity is $O(n_{\mathcal{C}}^3)$. The pruning stage requires calculation of differences between internal degrees and external degrees for each node in the final subpopulation *core* $\mathcal{C}_f$. It takes linear time of $O(n_{\mathcal{C}})$. Recalculating $k_{\mathcal{C}}^{int}$ and $k_{\mathcal{C}}^{ext}$ both have a complexity of $O(n_{\mathcal{C}}^2)$. So the total complexity of our T-CLAP algorithm is $O(n_{\mathcal{C}}^2(n_{\mathcal{C}} + n_{\mathcal{B}}))$, and $O(n_{\mathcal{C}}^3)$ in a sparse network.

4 Experiments

In this set of experiments, we compare the performances of T-CLAP, Newman's modularity maximization using vector partitioning and Clauset's greedy maximization of local modularity. We will call the latter two algorithms the Newman and Clauset algorithms, respectively. For the comparison between

T-CLAP and the Newman algorithm, we generally only compare the running time, because Newman's algorithm is a graph partitioning algorithm and does not return any subpopulations (beyond the clusters in the returned partition). For the comparison between T-CLAP and the Clauset algorithm, the comparison is based on the quality of the subpopulations returned, which is measured by the I-E ratio and density, plus speed, which is measured by running times. (The server we used for experiments has an Intel Xeon X5550 CPU at 2.67GHz with four cores and memory size at 24 GB.)

For each experiment we repetitively ran 100 sub-experiments. For each sub-experiment, we generate a random graph, which is also called the population, at a specific size and with a constant expected degree $\mathbb{E}(k_j)$ for each node. In our setting, we always keep the $\mathbb{E}(k_j)$, which is also $\mathbb{E}(k_j^{ext})$, as four. We then plant blocks in the population graph. Each node in the population graph belongs to one and only one block b, each b consists of 100 nodes. Thus all the populations created have multiple blocks in them. We then design two different kinds of population graphs. In the first kind, all the communities have a unique $\mathbb{E}(k_j^{int})$, and a node size of 100. Such a setting implies each block has a constant density $\delta(b)$, (see Figure 6). We also adjust the difference between the background graph density and block density. We increase the density of background graph and keep the block density as a constant. The smaller the difference between these two densities, more difficulty the algorithms have when extracting dense subpopulations. We want to know which algorithm performs better in terms of the measures chosen. In the second kind of population, blocks belong to four sets of $\mathbb{E}(k_j^{int})$[5]. The $\mathbb{E}(k_j^{int})$ in each group are are 8, 16, 24, and 32 respectively, (see Figure 7). We transform $\mathbb{E}(k_j^{int})$ to the density $\delta(b)$ for each block. The numbers of blocks in each expected degree set are all the same.

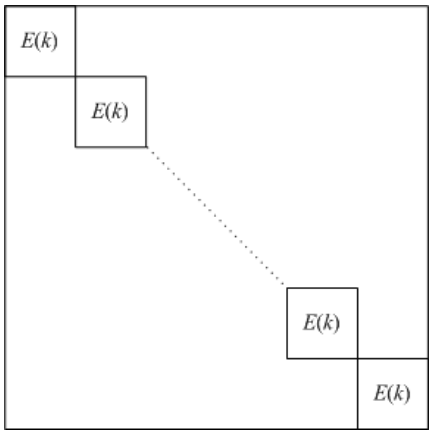


Figure 6: Block construction, constant expected internal degree blocks

[5] $\mathbb{E}(\max(I\text{-}E))=\{0, 0.33, 0.50, 0.60\}$

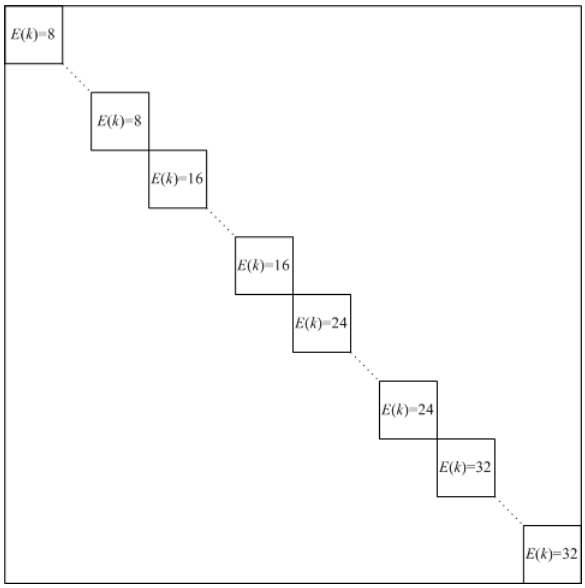


Figure 7: Block construction, Four sets of expected internal degree blocks

4.1 T-CLAP vs the Newman algorithm

In this experiment, we compare the performances of T-CLAP and Newman algorithms[6]. We generate networks with background graph sizes at 2000. We design two different sets of experiments based on the data generated. In the first set of experiments, the population networks on which we run the algorithms have 20 communities. All communities have the same size of 100, and each of them has constant expected internal degree, the total number of edges within the community. It is obvious that when the difference between the expected degree connecting to nodes within the community and expected degree connecting to outside of the community gets smaller, the more difficult it is for the algorithm to extract the community. So we first generate four sets of networks, each set has 100 populations at the size of 2000, with a constant expected degree of four. Then, in each set, we plant communities that each node has constant internal degree $\mathbb{E}(k_j^{int})$ at 8, 16, 24, and 32, respectively. In the second set of experiments, we generate 100 populations at the same size, with same numbers of communities. But instead of having a constant expected internal degree, the communities are grouped in four different classes. Each class has an assigned internal degrees. In each class, nodes in community have expected internal degree of 8, 16, 24 and 32. The populations on which both T-CLAP and the Newman algorithm run are the same.

4.1.1 Constant expected internal degree blocks

Since the Newman algorithm is a graph partitioning algorithm, it does not return subpopulations. It follows that we cannot obtain measure of subpopulation I-E ratios or densities. Instead, the I-E ratios we obtained for the Newman algorithm in this experiment is the one with the highest value across all communities in each population. Consequently the density is from the communities with the highest

[6]Newman kindly provided his modularity spectral maximization algorithm in C, while our T-CLAP algorithm is implemented in Python.

value. For the T-CLAP algorithm, we use the same stopping rule $s^{\mathcal{C}}$ at 150, seed set size n_{Θ} at 35, cycle of I-E recalculation r at 5 in all the experiments.

	T-CLAP				**Newman**			
	(1)	**(2)**	**(3)**	**(4)**	**(1)**	**(2)**	**(3)**	**(4)**
$\mathbb{E}(k_j^{int})$	8	16	24	32	8	16	24	32
$n_{\mathcal{S}}$	94	100	100	101	—	—	—	—
(*s.d.*)	(3.1)	(0.59)	(0.00)	(10.0)	—	—	—	—
max.	99	100	100	200	—	—	—	—
min.	83	97	100	100	—	—	—	—
$\mathbb{E}(\max(I\text{-}E))$	0.00	0.33	0.50	0.60	0.00	0.33	0.50	0.60
I-E	−0.0046	0.35	0.52	0.61	0.091	0.41	0.56	0.65
(*s.d.*)	(0.039)	(0.024)	(0.020)	(0.017)	(0.022)	(0.014)	(0.011)	(0.0095)
max.	0.082	0.42	0.56	0.66	0.16	0.45	0.59	0.67
min.	−0.095	0.29	0.47	0.57	0.050	0.37	0.54	0.62
CPU time (*sec.*)	2.2	1.6	1.9	2.2	155.7	156.3	159.4	156.4
(*s.d.*)	(0.58)	(0.41)	(0.46)	(0.45)	(0.98)	(1.1)	(4.0)	(0.62)
max.	3.6	3.3	3.5	5.3	158.4	162.4	173.2	157.8
min.	0.98	1.1	1.2	1.3	153.6	154.6	155.1	155.2
	$s_1^{\mathcal{C}} = s_f^{\mathcal{C}} = 150$, $r = 5$							

$n = 2000$, $\mathbb{E}(k_j^{ext}) = 4$, 100 runs

Table 2: T-CLAP VS the Newman algorithm, constant expected internal degree blocks

We found the Newman algorithm performs very well in getting the communities with the highest I-E ratio in each class of expected community internal degrees. The average highest community I-E ratio from the population with the expected community internal at eight is 0.091 – higher than that of T-CLAP at −0.0046. In the other classes, I-E ratio returned by the Newman algorithm is slightly better than those for T-CLAP. The densities of the subpopulations extracted by T-CLAP and communities with highest value I-E ratio by the Newman algorithm are very similar, with the Newman algorithm having a slight edge. However, a major difference emerges when we consider speed. On average, the running time of T-CLAP is between 1.6 and 2.2 seconds, while Newman algorithm is between 155.7 to 159.4 seconds. So, given the same population, T-CLAP is 70 to 98 times faster than the Newman algorithm.

4.1.2 Four sets of expected internal degree blocks

In real world networks, it is more likely that different communities have different expected internal degrees. In this set of experiments, each population has communities belonging to four different classes. All nodes in communities of the first class have an expected internal degree of 8 ($\mathbb{E}(k_j^{int}) = 8$); nodes in communities of the second class have an expected degree of 16 ($\mathbb{E}(k_j^{int}) = 16$); nodes in communities of the third class have an expected degree of 24 ($\mathbb{E}(k_j^{int}) = 24$); nodes in communities of the fourth class

have an expected degree of 32 ($\mathbb{E}(k_j^{int}) = 32$). The measures we use are average I-E ratio across all partitions, average density of all partitions, and running time (Table 3). When the population size is 2000, it takes 1.9 seconds on average for our T-CLAP algorithm to extraction a subpopulation at the size of 100, while it takes Newman's algorithm 166.0 seconds on average, 87 times higher than T-CLAP, to partition the network at the same size. We also ran experiments when the population size increases to 4000, it takes 3.2 seconds on average for our T-CLAP algorithm to extraction a subpopulation at the size of 100, while it takes Newman's algorithm 1284.2 seconds on average, 401 times higher than T-CLAP, to partition the network at the same size. The running time of Newman algorithm is polynomial to population size and cannot proceed because of the scale when population size is beyond 4000. In this experiment, we found Newman algorithm is not practical for extracting subpopulations for any network with size higher than 2000.

	T-CLAP		**Newman**	
	(1)	**(2)**	**(1)**	**(2)**
n	2000	4000	2000	4000
$\mathbb{E}(k_j^{ext})$	4	4	4	4
Runs	100	12	100	12
$n_{\mathcal{S}}$	100	100	—	—
(*s.d.*)	(0.00)	(0.00)	—	—
max.	100	100	—	—
min.	100	100	—	—
$\mathbb{E}(\max(I\text{-}E))$	0.60	0.60	0.60	0.60
I-E	0.61	0.60	0.63*	0.62*
(*s.d.*)	(0.022)	(0.014)	(0.024)	(0.0085)
max.	0.66	0.62	0.66	0.64
min.	0.51	0.57	0.60	0.61
CPU time (*sec.*)	1.9	3.2	166.0	1284.2
(*s.d.*)	(0.51)	(1.2)	(5.3)	(28.9)
max.	3.7	5.7	181.2	1339.1
min.	1.2	2.1	154.5	1245.8
	$s_1^{\mathcal{C}} = s_f^{\mathcal{C}} = 150$		* Maximum I-E of all partitions	
	$r = 5$			

$\mathbb{E}(k_j^{int}) = \{8, 16, 24, 32\}$

Table 3: T-CLAP VS the Newman algorithm, four sets of expected internal degree blocks

4.2 T-CLAP vs the Clauset algorithm

We now compare the performance of T-CLAP algorithm with that of the Clauset algorithm. The measures we use for the comparisons are I-E ratios and running times. In this experiment, the expected background degree is four, and the expected community internal degree is 16. Such data are more similar to real data. In reality, the number of connections within a social community is much higher than that of

a global community. The community size in each population is consistent at 100. Since the population size of each experiment population is 100,000, there are 1000 communities in each population. We again design two sets of experiments as those in the experiment in the last section. Communities have constant expected internal degree and communities in 4 classes of expected internal degree. The populations on which both T-CLAP and the Newman algorithm run are the same. Since both T-CLAP and the Clauset algorithm requires estimated community size as a parameter, we compare the algorithm in the situations when we underestimate, correctly estimate, and overestimate the community size in each set of experiment. We compare when stopping rule of T-CLAP is 67, 100, 150 respectively[7] , and correspondingly number of nodes to explore, n_k, as 67, 100 and 150 also. Since n_k has the same objective as stopping rule in T-CLAP, it is also referred as the stopping rule hereafter.

4.2.1 Constant expected internal degree blocks

When all the blocks have constant expected degree at 16, as shown in Table 4, T-CLAP has better I-E ratios and subpopulation densities. It does not matter if we set the stopping rule lower then, equal to, or higher than block size, T-CLAP can precisely return subpopulations with size at 101, almost the same as the planted blocks. When we underestimate subpopulation size, T-CLAP can include nodes left out in the stopping rule yet still belong to block through the clustering stage. When we overestimate subpopulation size, T-CLAP can exclude nodes not belonging to the block that were included through clustering stage through the pruning stage. The average I-E ratios of subpopulations returned by T-CLAP in three stopping rule situations are all at 0.60, while those returned by the Clauset algorithm are 0.0048, 0.58 and 0.16 respectively. The Clauset algorithm only has a comparable average I-E ratio at 0.58 when we correctly estimate block size; when we underestimate subpopulation size at 67, it returns subpopulations with much lower average I-E ratio at 0.0048; when we overestimate subpopulation size at 150, the Clauset algorithm returns subpopulations with a lower average I-E ratio at 0.16.

We also compare the running times of these two algorithms. The running time of the T-CLAP algorithm is slower than the Clauset algorithm when we underestimate and correctly estimate subpopulation size, while is faster than the Clauset algorithm when we overestimate subpopulation size. Given the fact in most real large social networks the block size is much larger than 100, T-CLAP tends to have better running time than the Clauset algorithm.

We plotted all I-E ratios returned by both algorithm in Figure 8. The x-axis is I-E ratio of subpopulations extracted by the Clauset algorithm; the y-axis is I-E ratio of subpopulations extracted by T-CLAP. The triangles are the I-E ratios returned by both algorithms when stopping rule is 67. In this case, we can see the I-E ratios returned by T-CLAP are consistently at about 0.60, while the I-E ratios returned by the Clauset algorithm are consistently at about 0. The Xs are the I-E ratios returned by both algorithms when stopping rule is 150. We can see the I-E ratios returned by the Clauset algorithm improved to about 0.20, but still lower than those returned by T-CLAP, which is consistently at about 0.60. The Clauset algorithm only has comparable returned I-E ratio when the stopping rule is at 100, which are marked as crosses. Still, T-CLAP has a slightly better I-E ratio returned than the Clauset

[7]The rationale for these choices is that 67 is 2/3 of 100 and 100 is 2/3 of 150.

	T-CLAP			Clauset		
	(1)	**(2)**	**(3)**	**(1)**	**(2)**	**(3)**
$s^{\mathcal{C}}$	67	100	150	67	100	150
n_{Θ}	17	25	35	—	—	—
$n_{\mathcal{S}}$	101	101	101	67	100	150
(*s.d.*)	(0.59)	(1.2)	(0.63)	—	—	—
max.	101	101	101	—	—	—
min.	98	100	100	—	—	—
$\mathbb{E}(\max(I\text{-}E))$	0.60	0.60	0.60	0.60	0.60	0.60
$I\text{-}E$	0.60	0.60	0.60	0.0048	0.58	0.16
(*s.d.*)	(0.020)	(0.031)	(0.019)	(0.015)	(0.018)	(0.041)
max.	0.64	0.64	0.64	0.035	0.63	0.24
min.	0.65	0.55	0.55	−0.066	0.53	0.061
CPU time (*sec.*)	2.9	4.1	7.6	1.4	3.3	11.8
(*s.d.*)	(0.61)	(0.78)	(2.4)	(0.13)	(0.28)	(1.2)
max.	4.7	6.2	25.1	1.9	4.0	15.2
min.	1.7	2.3	4.0	1.2	2.7	9.5
	$r = 5$					

$n = 100000$, $n_b = 100$, $\mathbb{E}(k_j^{ext}) = 4$

Table 4: T-CLAP VS the Clauset algorithm, constant expected internal degree blocks, $\mathbb{E}(k_j^{int}) = 32$

algorithm. T-CLAP clearly returned subpopulations with better quality in terms of I-E ratio.

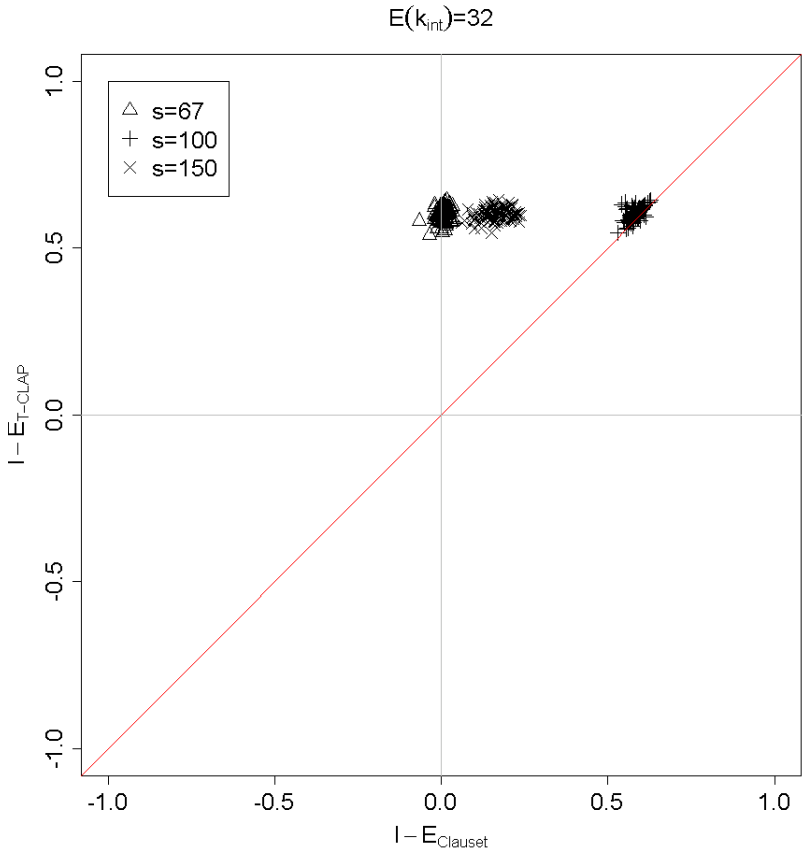


Figure 8: Comparison of average I-E ratio between T-CLAP and the Clauset algorithm

4.2.2 Four sets of expected internal degree blocks

In this set of experiments, each population has blocks belonging to four different classes, the same as the second set of experiments of T-CLAP and Newman algorithm comparisons. The same number of nodes are in four different communities, with expected internal degrees $\mathbb{E}(k_j^{int})$ set at 8, 16, 24 and 32. The measures we use are still the average I-E ratio, average density, and the average running time over 100 runs. The population size in this set of experiments is still at 100000, and the community size is also at 100. We keep all the parameters for T-CLAP the same as those of the experiment in Section 4.2.1. For stopping rules for T-CLAP $s_1^{\mathcal{C}}$ and $s_f^{\mathcal{C}}$ we again used 67, 100 and 150. To have a fair comparison, we used the same stopping rules for the Clauset algorithm. Again, the rationale for different values for these parameters is that we want to compare the performance in the situations when we underestimate the subpopulation size, correctly estimate the subpopulation size, and overestimate the subpopulation size. We pick starting nodes from each of four expected community internal degree classes. The number of starting nodes from these four classes are the same. We also use same starting node in each population

for both T-CLAP and the Clauset algorithm.

When the stopping rule is 67, T-CLAP has average an I-E ratio at 0.56, much higher than the -0.19 value for the Clauset algorithm and close to the true value of 0.6 (Table 5). T-CLAP has worse running time than Clauset algorithm though, 1.7 seconds versus 1.3 seconds. It shows that when users underestimate the size of community, T-CLAP still can extract subpopulations with a reasonably high I-E ratio, while the Clauset algorithm cannot if the starting node is not in a dense area of the network. When the stopping rule is 100, T-CLAP has an I-E ratio at 0.57, again much better than the Clauset algorithm's 0.053. The Clauset algorithm performs better when the stopping rule is the same as the community size. T-CLAP has an average running time at 2.8 seconds, a little faster than Clauset algorithm's 3.3 seconds. When both algorithms have stopping rules at 150, T-CLAP has much better I-E ratio than Clauset algorithm. Also, the T-CLAP algorithm has a faster running time in 6.5 seconds, 2.7 seconds less than that for the Clauset algorithm.

	T-CLAP			**Clauset**		
	(1)	**(2)**	**(3)**	**(1)**	**(2)**	**(3)**
$s^{\mathcal{C}}$	67	100	150	67	100	150
n_{Θ}	17	25	35	—	—	—
$n_{\mathcal{S}}$	100	101	101	67	100	150
(*s.d.*)	(1.2)	(0.64)	(0.60)	—	—	—
max.	101	101	101	—	—	—
min.	95	99	99	—	—	—
$\mathbb{E}(\max I\text{-}E)$	0.60	0.60	0.60	0.60	0.60	0.60
I-E	0.56	0.57	0.58	-0.19	0.053	-0.12
(*s.d.*)	(0.068)	(0.054)	(0.052)	(0.089)	(0.18)	(0.16)
max.	0.63	0.63	0.63	0.050	0.60	0.42
min.	0.25	0.34	0.34	-0.34	-0.11	-0.26
CPU time (*sec.*)	1.7	2.8	6.5	1.3	3.3	9.2
(*s.d.*)	(0.67)	(1.3)	(5.1)	(0.15)	(0.29)	(0.56)
max.	6.9	8.8	35.2	1.7	3.9	10.6
min.	0.88	1.4	2.7	1.0	2.5	7.9
	$r = 5$					

$n = 100000$, $n_b = 100$, $\mathbb{E}(k_j^{ext}) = 4$, $\mathbb{E}(k_j^{int}) = \{8, 16, 24, 32\}$

Table 5: T-CLAP VS the Clauset algorithm, four sets of expected internal degree blocks

The I-E ratios returned by both algorithms are plotted in Figure 9. The three figures from left to right are for stopping rules at 67, 100 and 150 respectively. As in the last section, the x-axis is the I-E ratio of subpopulations extracted by the Clauset algorithm and the y-axis is the I-E ratio of subpopulations extracted by T-CLAP. Triangles represent 25 experiments that the starting nodes are in the communities with $\mathbb{E}(k_j^{int})$ at 8; crosses represent experiments that the starting nodes are in the communities with

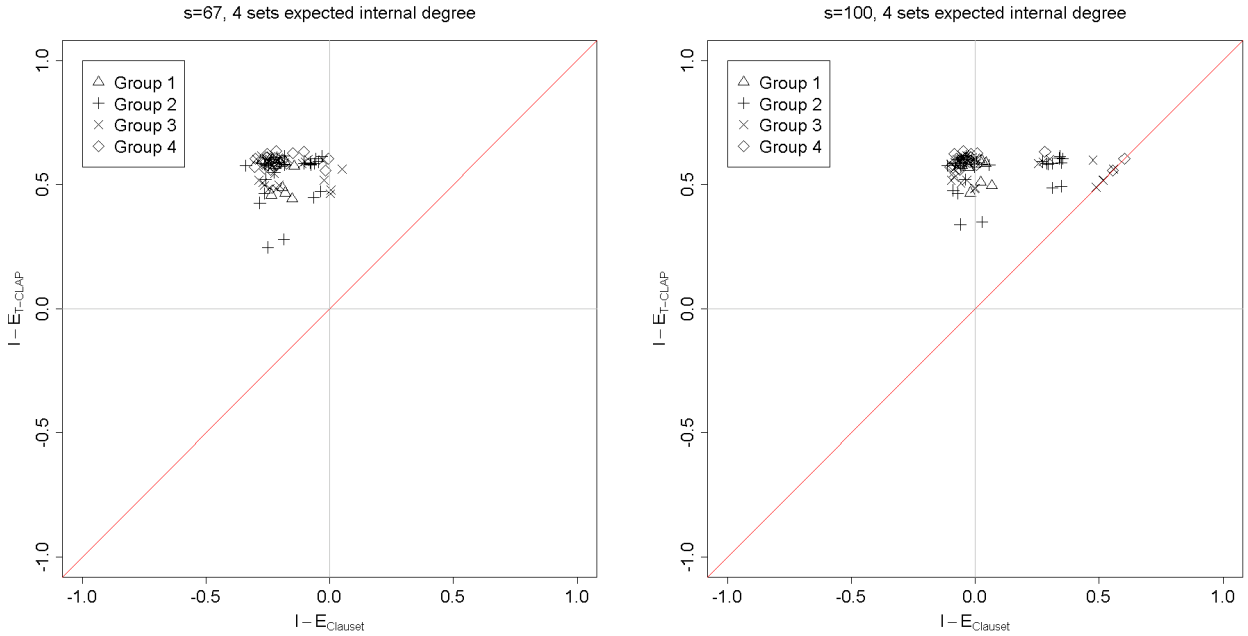
$\mathbb{E}(k_j^{int})$ at 16; Xs represent experiments where the starting nodes are in the communities with $\mathbb{E}(k_j^{int})$ at 24; and rhombuses represent experiments that the starting nodes are in the communities with $\mathbb{E}(k_j^{int})$ at 32; In these three figures, we can see the nodes are all at or above the diagonal line, which shows T-CLAP extract subpopulations with higher I-E ratios than the Clauset algorithm in all stopping rule cases and expected community internal degree cases. When the stopping rule is 67, subpopulations extracted by T-CLAP mostly have I-E ratios at about 0.60, no matter what expected internal degree community the algorithm started. (Only two experiments are outliers, returning subpopulations with I-E ratios at about 0.25.) On the other hand, the subpopulations returned by the Clauset algorithm have I-E ratios below 0, which suggests all the subpopulations have boundary leakage problems. When the stopping rule is 100, T-CLAP still extracted subpopulations with a I-E ratio higher than 0.50. The Clauset algorithm performed better than the last stopping rule set but, even so, the majority of I-E ratios returned are still around 0, with only four experiments tied. When the stopping rule is 150, T-CLAP extract subpopulations with higher I-E ratios than Clauset in all experiments at values close to 0.60.

The scatter plot for algorithm parameters and the measures we used to evaluate subpopulation quality are shown in Figure 10. Plots in the first row describe the correlation of the expected internal degree class of 100 runs and all the other variables. Class 1 are communities with expected internal degree $\mathbb{E}(k_j^{int})$ at 8, class 2 are the communities with $\mathbb{E}(k_j^{int})$ at 16, class 3 are the communities with $\mathbb{E}(k_j^{int})$ at 24, and class 4 are communities with $\mathbb{E}(k_j^{int})$ at 32. Plots in the second row and second column describe the correlation between the size of subpopulations extracted by T-CLAP algorithm and all the other variables. Plots in the third row and third column describe the relationship between the I-E ratio of subpopulations extracted by the T-CLAP algorithm and all the other variables. Plots in the fourth row and fourth column describes the relationship between the running time of 100 T-CLAP algorithm runs and all the other variables. Plots in the fifth row and fifth column describe the relationship between the I-E ratio of subpopulations extracted by the Clauset algorithm and all the other variables. Plots in the sixth column describes the relationship between the running time of 100 Clauset algorithm runs and all the other variables.

Plots in row 1 and column 3 shows the relationship between I-E ratios of T-CLAP and the expected degree set of the starting node. We found no matter in which set we start the extraction, T-CLAP can always ends up with finding a subpopulation with an I-E ratio higher than 0.50. In contrast, the Clauset algorithm does not perform well no matter where the algorithm starts. The only time in which the Clauset algorithm can extract subpopulations with I-E ratio higher than 0.20 is by starting in a community with expected internal degree larger than 8. Also, we find the running time of T-CLAP does not change much with the size of subpopulations returned (shown in the plot in row 1 column 4).

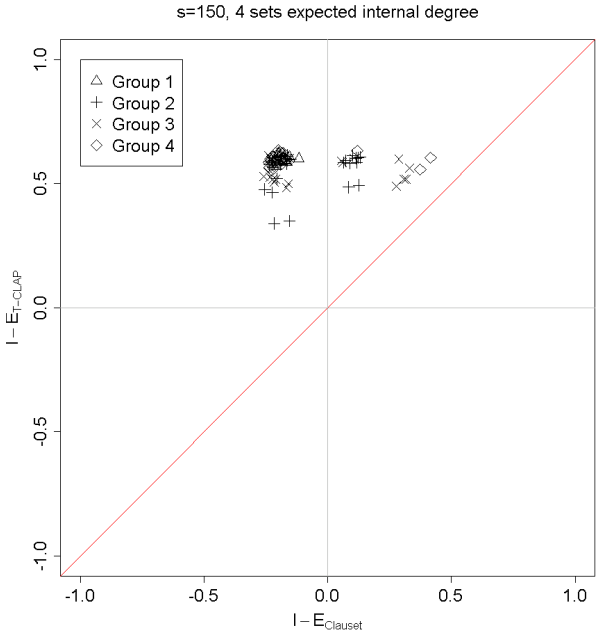
4.3 Summary of Simulations

A comparison of all three algorithms performances, measured by running time and I-E ratio is depicted in Figure 11. In this comparison, all experiments are done using the same 100 populations at the size of 2000. Every node belongs to one community, and all communities have the same sizes of 100. The stopping rules for T-CLAP and the Clauset algorithm are both set at 150. The x-axis is running time,



(a) $s^{\mathcal{C}} = 67$

(b) $s^{\mathcal{C}} = 100$



(c) $s^{\mathcal{C}} = 150$

Figure 9: Scatter plots at different subpopulation size: T-CLAP verses the Clauset algorithm

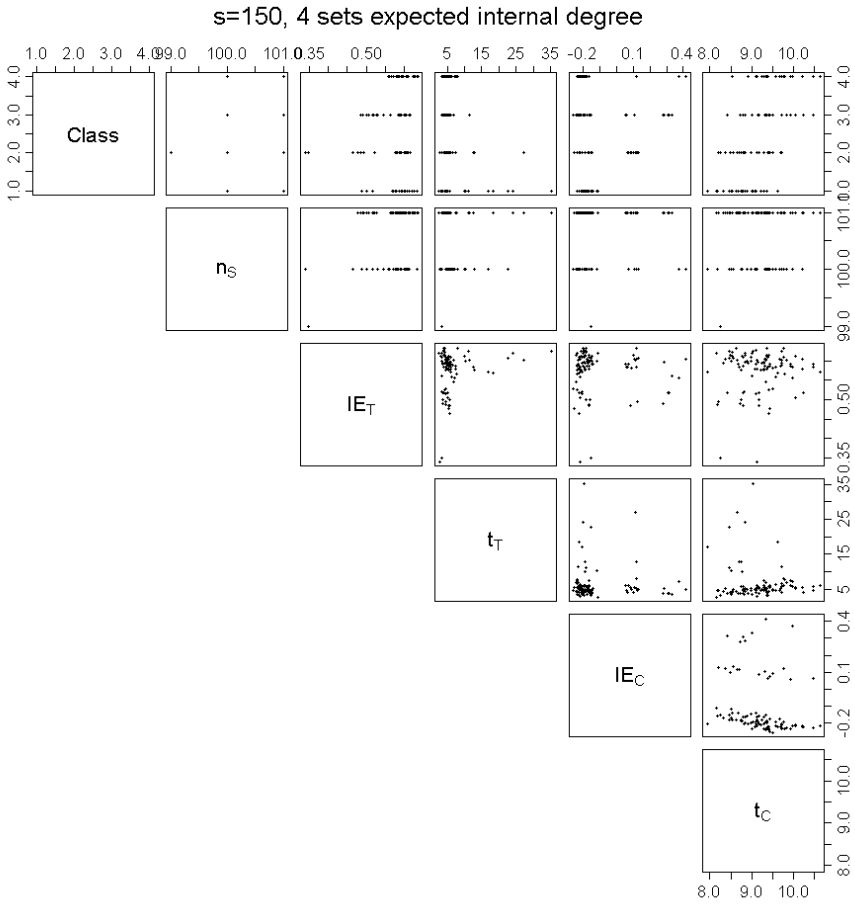


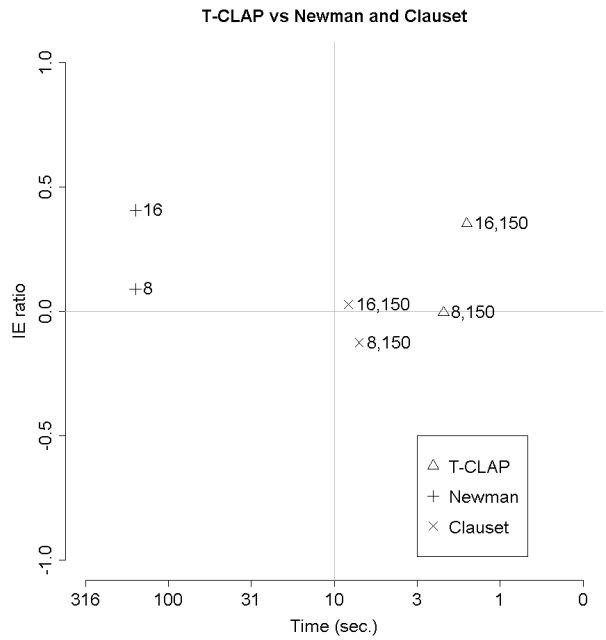
Figure 10: Correlation graph with the stopping rule set at 150

where the small value is the right side and the large value is at the left side. The y-axis is the I-E ratio, where $+1$ means all edges in the subpopulations are internal edges, while -1 stands for all edges are external edges. So the data points in the upper right part of the figure have faster running times and higher I-E ratios. We use triangles to represent results from the T-CLAP algorithm, crosses to represent results from the Newman algorithm, and Xs to represent results from the Clauset algorithm. Figure 11(a) shows the result of comparisons when the expected internal degree of block are 8 and 16. The results from the Newman algorithm are all at the upper left part of the graph (quardrant II), meaning this algorithm can identify the most dense community by partitioning, implicitly returning the subpopulation with the highest I-E ratio in the population. We also observe that as the difference between the expected degree of background graph and expected degree of community internal degree increases, the Newman algorithm can identify subpopulations with higher I-E ratios. However the time cost is much higher. It takes the algorithm more than 150 seconds to identify subpopulations at the size of 100 within a population at the size of 2000. The results from the Clauset algorithm are around the center (origin) of the graph, meaning the running time is about 10 seconds, and I-E ratios are around 0. The Clauset algorithm is 10 times faster than Newman algorithm, but the I-E ratio is much lower, about 0.4 to 0.5 lower, than thse from the Newman algorithm. The results from the T-CLAP algorithm are at the upper right part of the figure, except for the case when the expected internal degree of community is 4. In this case, the internal degree of each node is 4, whereas the external degree is 4 also. So on average the I-E ratio is about 0. The subpopulations returned by our algorithm have comparable I-E ratios to those of the Newman algorithm and higher than those of the Clauset algorithm. Our algorithm is the fastest among all three algorithms. All experiments have an average running between 1 and 3 seconds. In summary, T-CLAP can return subpopulations with I-E ratio that are close to the highest in the population, and can finish this extraction much more quickly than the other two algorithms – 3 to 5 times faster than the Clauset algorithm and 50 to 100 times faster than the Newman algorithm.

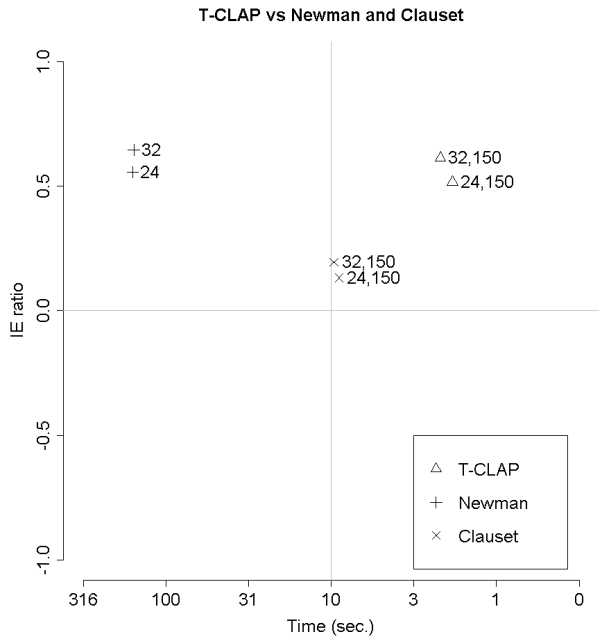
4.4 Cellular Phone Data

We then compare the performance of T-CLAP and the Clauset algorithm using real data with node size at the order of millions. The Newman algorithm is not considered here since it cannot handle data at this size. The data were obtained from a large Indian telecommunications company (source and raw data confidential). We have cellular phone call records over a three-month period. The original data set contains approximately 26 million unique users. Since we have both the hash values of phone number for the calling party and the called party, the network we have is a directed one, where calling party is the source node and called party is the terminal node, and the phone call is the edge between the two parties. There were about 1 billion phone calls initially in our data set. It includes all the phone calls received by the company's customers in three months. Those phone calls were initiated by customers both inside and outside of the company. Our analysis is only constrained to phone calls that occur between customers using the same provider. As a result, we have some of their demographic information such as age and gender for future researches.

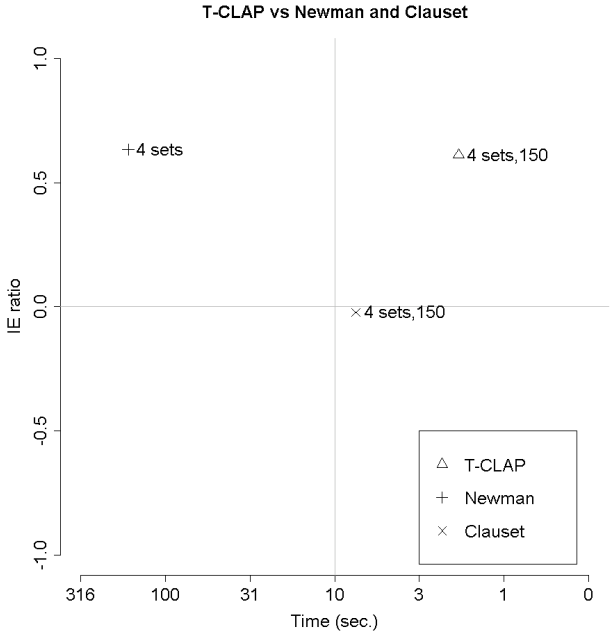
Identifying reciprocated calls is important in our data set. Since the phone call social network is directed, asymmetry can exist between callers. Any asymmetric connection between two callers may



(a) $\mathbb{E}(k_j^{int}) = 8, 16$



(b) $\mathbb{E}(k_j^{int}) = 24, 32$



(c) 4 sets of $\mathbb{E}(k_j^{int})$

Figure 11: Comparison of performance for the T-CLAP, Newman and Clauset algorithms

indicate a weak connection, and may be less likely to indicate a relationship that provides social influence of either party on the other. An asymmetric connection indicates an unstable relationship while symmetric connections imply equal and stable connections (Hanneman and Riddle, 2005). We define reciprocity for dyads (A, B) as the condition in which A calls B and B calls A in same calendar month. We interpret reciprocity as an increase probability that the two parties are acquaintances. Thus we further constrain our analysis to include only the data that involve reciprocal dyads. Constrained by these requirements, the size of our data becomes to about 1.4 million nodes and 15 million edges.

In this set of experiments, we used both T-CLAP and the Clauset algorithm to extract 20 subpopulations from the population at the size of 1.4 million. Each T-CLAP experiment started with a unique random seed. The stopping rule and seed set size we used for all 20 T-CLAP runs are constantly at 50 and 15, respectively. After extracted a subpopulation using T-CLAP, we used the same seed, and the subpopulation size returned by T-CLAP as the stopping rule n_k, for the Clauset algorithm. So in each run, both algorithms had the same size of subpopulation extracted. The average size of subpopulations returned is 153, while the maximum size is 515 and the minimum size is 50.

The average I-E ratio returned by T-CLAP is at 0.16, much better than that of the Clauset at -0.16. We found that given the same seeds in the experiments, T-CLAP always return subpopulations with positive I-E ratio. Those ratios returned by T-CLAP are at least at 0.0030, and the maximum is at 0.62 (Table 6). Such results are much better than those of the Clauset algorithm – almost all the subpopulations extracted by Clauset have negative I-E ratio, except only one subpopulation has a positive ratio at 0.04. Considering that any subpopulation with a negative I-E has more edges to the external network than to the internal is not self-contained, almost all the subpopulations extracted by Clauset are not actually subpopulation, and are not useful for any social network analysis.

The average running time of T-CLAP is at 10.1 seconds, faster than the Clauset algorithm at 13.3 seconds. We also found from Table 6 when subpopulation is at the minimum size of 50, T-CLAP has a running time at 1.5 seconds, slower than that of the Clauset algorithm at 0.29 seconds; when the returned subpopulation is at the maximum of 515, T-CLAP has a running time at 53.3 seconds, much faster than that of Clauset at 115.3 seconds. We are aware that when the size of returned subpopulations is small, *e.g.* below 200, Clauset algorithm is faster than T-CLAP, while when the returned subpopulation sizes are beyond 200, the Clauset algorithm becomes slower than T-CLAP. The running time of the Clauset algorithm increases much faster than that of T-CLAP when the size of subpopulation returned increases.

So, given the real data at node size of more then 1 million, T-CLAP performs much better than the Clauset algorithm. The subpopulation I-E ratios returned by T-CLAP are much higher than those returned by the Clauset algorithm, and T-CLAP is faster. Furthermore only these subpopulations returned by T-CLAP have all positive I-E ratios. Although the Clauset algorithm is faster when subpopulation size is small, almost all the subpopulations returned by Clauset have boundary leakage problem and cannot be used for any social network analysis models. Also if we want to extract subpopulation with large size, e.g. larger than 1000, T-CLAP are much more practical than the Clauset algorithm.

	T-CLAP	Clauset
$s^{\mathcal{C}}$	50	$n_{\mathcal{S}}$
n_Θ	15	—
$n_{\mathcal{S}}$	153	153
(*s.d.*)	(125.8)	(125.8)
max.	515	515
min.	50	50
I-E	0.16	−0.16
(*s.d.*)	(0.19)	(0.11)
max.	0.62	0.040
min.	0.0030	−0.36
CPU time (*sec.*)	10.1	13.3
(*s.d.*)	(13.6)	(29.4)
max.	53.3	115.3
min.	1.5	0.29
	$r = 20$	
	$n = 1459800$, 20 runs	

Table 6: T-CLAP VS the Clauset algorithm, cellular phone call data

5 Conclusion

One large challenge for social network research is to make analyses tractable for larger networks that those traditionally considered. Both the number of actors and connections can be in the millions. It is impossible to analyze the effects of the whole network given most of currently available computing power. One way to solve this problem is by analyzing subpopulations extracted from a global network. Also, if there are genuinely different communities in the whole network, it is desirable to extract subpopulations in the same operational way. In the case of scientific networks, extracting subpopulations from the network structure rather than imposing arbitrary definition of scientific disciplines seems much more preferable. However, subpopulation or community extraction is not a trivial problem. It has been long known that there are many open questions in extracting communities from networks. The most significant is performance. The complexity of many of the community detection methods using modularity as quality function can reach $O(nm)$, where n and m are the numbers of actors and connections in the network, respectively. We designed an innovative algorithm to extract local communities (subpopulations) from large scale network. Our method does not require any parameters about the network. It has a complexity of $O(n_{\mathcal{C}}^2(n_{\mathcal{C}} + n_{\mathcal{B}})$, where $n_{\mathcal{C}}$ is the size of the *core*, and $n_{\mathcal{B}}$ is the size of boundary nodes. So if the size of the subpopulations we want to extract is much smaller than the whole population (for example 1000 versus 1 million), our method has a complexity of $O(10^7)$, compared to modularity spectral optimization's $O(10^{13})$. Given the fact that more and more social networks data are large scale, our method provides a viable solution for extracting internally densely connected subpopulations from large social networks. Although greedy maximization local modularity has a same complexity at $O(10^7)$ if n_k is at the order of 10^3, and average degree of each node is at 10. But its actually running time

is still slower than T-CLAP's. T-CLAP returns subpopulations with quality, measured by I-E ratio, comparable to Newman, and higher than Clauset. It is almost a hundred times faster than Newman algorithm and several times faster than Clauset. T-CLAP can extract subpopulations with much higher I-E ratios than the Clauset algorithm, and these ratios are very close to the highest I-E ratios among all blocks. We found similar results in both experiments using simulated and real data.

References

Batagelj, V. and Mrvar, A. (1998). Pajek – a program for large network analysis. *Connections*, 21(2):47–57.

Clauset, A. (2005). Finding local community structure in networks. *Physical Review E*, 72(2):026132.

Coleman, J. S. (1958). Relational analysis: The study of social organization with survey methods. *Human Organization*, 17(4):28–36.

Girvan, M. and Newman, M. E. J. (2002). Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826.

Goodman, L. A. (1961). Snowball sampling. *The Annals of Mathematical Statistics*, 32(1):148–170.

Hanneman, R. and Riddle, M. (2005). *Introduction to social network methods*. Online.

Krackhardt, D. and Stern, R. N. (1988). Informal networks and organizational crises: An experimental simulation. *Social Psychology Quarterly*, 51(2):123–140.

Newman, M. E. J. (2004). Detecting community structure in networks. *The European Physical Journal B – Condensed Matter and Complex Systems*, 38(2):321–330.

Newman, M. E. J. (2006a). Finding community structure in networks using the eigenvectors of matrices. *Physics Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 74(3):036104.

Newman, M. E. J. (2006b). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582.

Salganik, M. J. and Heckathorn, D. D. (2004). Sampling and Estimation in Hidden Population Using respondent-Driven Sampling. *Socialogical Methodology*, 34:193–239.

Semaan, S., Lauby, J., and Liebman, J. (2002). Street and Network Sampling in Evaluation Studies of HIV Risk-Reduction Interventions. *AIDS Review*, 4:213–223.

Snijders, T. A. B. (1992). Estimation on the basis of snowball samples: How to weight? *Bulletin de Methodologie Sociologique*, 36(1):59–70.