An Extremely Fast Algorithm for Identifying High Closeness Centrality Vertices in Large-Scale Networks

Vladimir Ufimtsev  
Computer Science Department  
University of Nebraska-Omaha  
vufimtsev@unomaha.edu

Sanjukta Bhowmick  
Computer Science Department  
University of Nebraska-Omaha  
sbhowmick@unomaha.edu

Abstract—The significance of an entity in a network is generally given by the centrality value of its vertex. For most analysis purposes, only the high ranked vertices are required. However, most algorithms calculate the centrality values of all the vertices. We present an extremely fast and scalable algorithm for identifying the high closeness centrality vertices, using group testing. We show that our approach is significantly faster (best-case over 50 times, worst-case over 7 times) than the currently used methods. We can also use group testing to identify networks that are sensitive to edge perturbation.

I. INTRODUCTION

Analysis of networks has become a key tool for studying systems of interacting entities such as, those arising in biology, social-sciences and epidemiology. An important objective in studying such systems is to identify key vertices in the system. In network terms, this translates to identifying high centrality vertices. For analysis purposes we only need to know the identity of the top high ranked vertices. Moreover, unless the top-ranked vertices are significantly higher than the other vertices in the system, their significance diminished. Indeed, if the centrality values of most vertices are close, this indicates that the network will be sensitive to edge perturbation, i.e. the ranking will change with a slight fluctuation of the network connectivity.

In this paper, we develop a highly scalable group testing based algorithm for finding high ranked closeness centrality (CC) vertices. We demonstrate that a massive reduction (as much as 50 times) in time can be achieved by our algorithm as compared to the currently used methods. Our algorithm is not based on any approximation method or graph sampling, the speed is solely due to its formulation as a group testing process.

The only time our method fails to find high ranked vertices is when a large portion of the vertices have close CC values and therefore cannot be considered to be the more significant value in the group. These "negative" results are also helpful, as they indicate that the CC values of the network is sensitive under edge perturbation.

II. BACKGROUND

A network is defined by a pair of sets \( G = (V, E) \), where \( V \) is the set of vertices and \( E \) is the set of edges. The centrality of a vertex is a measure which specifies the importance of the vertex in a network. The closeness centrality (CC) of vertex \( v \) is defined as: \( CC(v) = \frac{1}{\sum_{s \in V} d(v, s)} \), where \( d(v, s) \) is the distance between \( v \) and \( s \). Vertices that are overall closer to other vertices in the network will have higher CC values.

Related Work. A scalable parallel algorithm for closeness centrality proposed in [2] involves computing a breadth first search (BFS) with each vertex as a root. With \( p \) processors, the time taken to compute the closeness centrality of all the \( |V| \) vertices would be \( O\left(\frac{|V|}{p}\right) + |E| \). Many parallel network analysis libraries [4], [9], [6] also contain algorithms for computing closeness centrality. An alternative is to use approximate computations such as those proposed by Eppstein and Wang [5]. Recently, Sariyuce et. al. presented a fast algorithm for updating closeness centrality on dynamic networks [11]. Okamoto et. al. [10] ranks of \( k \) highest closeness centrality vertices using a hybrid of approximate and exact algorithms.

III. GROUP TESTING FOR IDENTIFYING HIGH CLOSENESS CENTRALITY VERTICES

Group testing (GT) is a mathematical technique commonly employed in the design of screening experiments, with the objective of finding a specified number of special valued units from a large population with the fewest tests.

We have used a superimposed code constructed from a Latin Square to apply group testing on networks for finding high closeness centrality vertices. This method is guaranteed to find 2 special units. Given a population of \( n \) units, we take the set of contiguous integers \( \{1, 2, ..., l\} \), where \( l = \lceil\sqrt{n}\rceil \). We then create an \( l \times l \) Latin square, which is a matrix \( A \), where \( A_{i,j} \in \{1, 2, ..., l\} \) such that each element from \( \{1, 2, ..., l\} \) appears exactly once in any given row and column (left diagram in Figure 1(a)).

We construct a superimposed code \( X \) from the Latin square \( L \) as follows; the first 2 positions in any given column in \( X \) are coordinates (row and column) in \( L \) and the 3rd position is the element in \( L \) at those coordinates (middle diagram in Figure 1(a)). We then encode each integer in \( X \) in its binary form to get the superimposed code. Each integer is coded as a binary vector of length \( l \), where for integer \( i \), the vector has zero in all positions, except at position \( i \) which has one. In other words, the binary representation of integer \( i \) is the \( i^{th} \).
row (or column) of an $l \times l$ identity matrix (right hand diagram in Figure 1(a)). As an example consider the eighth column in the middle diagram of Figure 1(a). The row value (top row) is 3 and the value (last row) is $\lfloor \frac{|V|}{l} \rfloor$. Therefore the minimum weight is $w = 3$ and maximum intersection is $\lambda = 1$. Based on the Kautz-Singleton Bound [7] the guaranteed value for the strength parameter $d \geq \frac{\sqrt{\frac{|V|}{l}}}{\lambda} = \frac{\sqrt{|V|}}{2} = 2$. Thus the Latin square guarantees finding 2 defective units from the population.

The process of finding closeness centrality vertices using group testing consists of three major steps as follows;

**Step 1. Creating Appropriate Groups of Vertices.** The coding matrix will contain size $3\sqrt{|V|}$ groups to be tested and each group will contain $\sqrt{|V|}$ vertices. We do not create and store this matrix explicitly. Instead, based on the number of vertices and the group number we can identify the vertices that will be part of the group. This information is used in the second step to compute the centrality per group of vertices. The time required for this step is proportional to the number of groups, and is $O(3\sqrt{|V|})$.

**Step 2. Computing Centrality of the Groups.** The vertices in each group are tested together to form a supervertex and then the centrality is computed for the supervertex in each group. We do not explicitly combine the vertices. Instead, we execute the BFS such that the root consists of multiple sources. Thus the execution time is $O(|V| + |E|)$ per group and the total time for this step is $O(3\sqrt{|V|}(|V| + |E|))$.

**Step 3. Identifying High Ranked Vertices.** Each group produces a closeness centrality value for its corresponding supervertex. Once the centrality values for all the groups are obtained, we select a threshold such that the groups with centrality values above or equal to the threshold are marked.

---

**Algorithm 1** Identifying High Closeness Centrality Vertices.

*Input:* A connected graph $G = (V, E)$. *Output:* Vertices with High Closeness Centrality.

1. $R = 3\sqrt{|V|}$ \hspace{1cm} \triangleright Get number of groups
2. $MaxB = 5$ \hspace{1cm} \triangleright User Defined Bound on the Maximum Number of High Rank Vertices to be Obtained
3. for $i = 1 : R$ do in parallel \hspace{1cm} \triangleright Compute centrality per group of vertices
4. \hspace{1cm} $Q_i \leftarrow \text{FindVertices}(i, |V|, Q_i)$ \hspace{1cm} \triangleright Get the vertices in each group
5. \hspace{1cm} $CC_i \leftarrow \text{MultiRootCC}(G, Q_i)$ \hspace{1cm} \triangleright Compute CC in each group
6. Gather all CC values to Root Processor
7. $T \leftarrow$ Third Highest $CC$ \hspace{1cm} \triangleright Set the threshold. Here we select the third highest CC of the supervertices.
8. while $T > 1$ do
9. \hspace{1cm} Broadcast $T$ to all processors
10. for $j = 1 : |V|$ do \hspace{1cm} \triangleright Initialize vertices as high rank is true
11. \hspace{1cm} $HighRank_j \leftarrow \text{TRUE}$
12. for $i = 1 : R$ do in parallel \hspace{1cm} \triangleright Mark off Vertices
13. \hspace{1cm} if $CC_i < T$ then \hspace{1cm} \triangleright Mark off Vertices
14. \hspace{1cm} \hspace{1cm} for all Vertex $j$ in Group $i$ do \hspace{1cm} \triangleright Mark off Vertices
15. \hspace{14cm} $HighRank_j = \text{FALSE}$
16. \hspace{1cm} Gather $HighRank$ values
17. \hspace{1cm} $high = \text{Number of Vertices marked as TRUE}$
18. if $1 < high \leq MaxB$ then
19. \hspace{1cm} Return the high ranked vertices
20. else
21. \hspace{1cm} if $high > MaxB$ then
22. \hspace{1cm} \hspace{1cm} Return Error Message: Too Many Vertices Found
23. else $T = T/2$ \hspace{1cm} \triangleright Else reduce threshold value

1. $Q \leftarrow \text{MultirootCC}(G, Q_i)$ \hspace{1cm} \triangleright Compute centrality of each group
2. for $v \in |V|$ do \hspace{1cm} \triangleright Compute centrality of each group
3. \hspace{1cm} $dist[v] = 0$
4. if $v \in Q$ then \hspace{1cm} \triangleright Compute centrality of each group
5. \hspace{1cm} $Visited[v] = \text{TRUE}$
6. else
7. \hspace{1cm} $Visited[v] = \text{FALSE}$
8. while Q is not empty do
9. \hspace{1cm} $Q1 = \emptyset$
10. for all $q \in Q$ do
11. \hspace{1cm} if $n$ is neighbor of $q$ and $Visited[n] = \text{FALSE}$ then
12. \hspace{1cm} $Visited[n] = \text{TRUE}$
13. \hspace{1cm} $Q1 \leftarrow Q1 \cup n$
14. \hspace{1cm} $dist[n] = dist[q] + 1$
15. \hspace{1cm} $CC \leftarrow CC + dist[n]$
16. \hspace{1cm} $Q = Q1$
17. $CC \leftarrow 1/CC$
as one and the rest are marked as zero.

To cover the binary vectors, we start with a set \( S \) of all vertices. Each time the closeness centrality of a group falls below the threshold, we remove the vertices forming the group from the set, \( S \). When all the groups have been considered, then the only vertices remaining in \( S \) will be the ones whose corresponding binary vector is covered by the resultant vector. These are the high centrality vertices. This process is done per group and each element in the group is considered for removal or keeping. Therefore in the worst case the time complexity for this step is \( O(3\sqrt{|V|} \times \lceil \sqrt{|V|} \rceil) \).

Selecting the appropriate threshold is important in obtaining the correct results. In our algorithm, we start with an initial threshold. If using this threshold does not provide sufficient number of high ranked vertices, we iteratively reduce the threshold value until about 2-5 high ranked vertices remain. While the Latin square method gives a strength of 2, by decreasing the threshold we start to cover more columns. In many cases these extra columns also correspond to the next highest closeness valued vertices.

Parallel Implementation. Steps 1 and 2 consist of independent operations are therefore easily parallelizable. Once the centrality results are computed, their values are compared using a reduction operation to obtain the threshold. Once the threshold is selected, the process of identifying which vertices remain can also be done in parallel. We mark the vertices to be deleted, and whatever remains are the high ranked ones. The pseudocode is given in Algorithm 1.

Given \( p \) processors, the total computation time taken therefore is \( \frac{1}{p}O(3\sqrt{|V|} \times (1+|V|+|E|+\lceil \sqrt{|V|} \rceil)) \) which can be approximated to \( O(\frac{\sqrt{|V|}}{p}((|V|+|E|))) \). The computational complexity of the standard parallel closeness centrality algorithm is \( O(|V|/p(|V| + |E|))) \), about \( \lceil \sqrt{|V|} \rceil \) times higher. The communication is only due to the reduction operations to send in the threshold values and to mark the vertices that are not high ranked and its complexity is \( O(3\sqrt{|V|})\log p) \).

Identifying False Positive Results. If the values of all the vertices are very close, then even a strict threshold identifies a very large number of vertices to be high ranked, which is an obvious indicator of false positives. Hence we put a bound to the number of vertices that can be reasonable marked as high rank. If the output exceeds the bound, we judge the answer to be incorrect. We posit that the false positives actually provide a deeper insight to the network. Mere ranking of vertices is meaningless unless the difference between the top ranked vertices and the ones below is significant. In such cases the ranking is more academic than of practical use.

Efficiency of Group Testing. For group testing to be efficient we should have an algorithm that is capable of computing the values per vertex. Let the average time for computing the centrality of a group of vertices be \( T_g \) and the average time for computing the centrality of one vertex be \( T_1 \). If we ignore the time to sort the values, then, in the Latin square case, group testing is efficient when \( \frac{T_g}{T_1} < \frac{\sqrt{|V|}}{3} \). In the case of closeness centrality, because we start from multiple sources at the same time \( T_g \) is nearly equal to \( T_1 \), irrespective of the group size. Therefore group testing is very efficient for this metric.

IV. Experimental Results

Our experiments were performed on a set of 4 real world networks (obtained from the Stanford Network Analysis Project [8]) and 4 synthetic networks obtained using the R-MAT generator [3]. We created two R-MAT structures (B1: \( a=.55, b=.15, c=.15, d=.15 \) ) and (B4: \( a=.65, b=.10, c=.10, d=.15 \) ). These values determine the structure of the network. Table II provides an overview of the networks.

![Scalability of GT for Closeness on RMAT Networks](image)

**Fig. 2.** Scalability of the Group Testing Algorithms. **Left Figure:** Real World Networks. **Right Figure:** R-MAT Networks.

Execution Time and Scalability We ran the experiments on the Tusker cluster at the Holland Computing Center at UNO which consists of 106 AMD Interlagos-based nodes (6784 cores) interconnected with Mellanox QDR Infiniband with 256 GB per node. We used distributed memory (MPI) as the programming paradigm. We used one core per node to ensure that the communication time is nearly equal between the processors. The execution times are given in Table I.

Group testing method is significantly faster than the parallel closeness centrality computation (best case 50 times, worst case over 7 times). The improvement more for larger networks because group testing improves about \( O(\sqrt{|V|}) \) times over the parallel implementation. The algorithm is also extremely scalable. Figure 2 shows the scalability of the R-MAT graphs.

Accuracy of Results. Table II provides the threshold and whether the identified vertices were indeed high rank. Although Latin square guarantees the 2 highest elements, in cases where the closeness centrality of the second highest vertex was not as significantly high from the other vertices, we obtained only the first. Conversely, sometimes we obtained the 1st, 2nd and the 3rd highest vertices. The only major failure was that of the Gnutella network. We only found low centrality vertices in this cases. This also indicates that compared to other networks Gnutella should be sensitive to perturbations.

Sensitivity of Networks. Sensitivity of network parameters under slight perturbations is an important concern because real-world networks are inherently noisy. A network model is stable if under slight changes to its connectivity, the parameters of the network do not vary significantly. To perturb the network we used the perturbation model developed in [1]. In this model, for a given parameter \( \epsilon \), \( 0 \leq \epsilon \leq |V| \), a present (absent) edge can be deleted (added) with a probability of \( \frac{\epsilon}{|V|} \).

We first find the top 10 ranked vertices in the network using a standard closeness centrality algorithm. Then we perturb
the network at various values of ϵ (0.05, 0.1, 0.25 , 0.5, 1) and obtain the top 10 CC vertices for the new networks. We measure the similarity of the original rank and the ranking from the perturbed networks using the Jaccard index. For two sets that are identical, the Jaccard index is 1 and for two disjoint sets the value is 0. The closer the index is to 1 the lower the effect of the perturbation.

As shown in Figure 3 the Jaccard index of Gnutella is always zero for all the values of ϵ. This indicates that the even for very small perturbations, the ids of the high ranked vertices completely change. In contrast, for the lower values of epsilon, the Jaccard index corresponding to the Brightkite network is above 50%, indicating that at least half of the top-ranked vertices in the original network retained their rank after perturbation. This is also borne out by the group testing results where our method can identify the top nodes in Brightkite.

V. DISCUSSION AND FUTURE WORK

In this paper, we presented a parallel group testing algorithm for finding the top-2 high closeness centrality vertices. Our method significantly outperforms the standard parallel algorithm for computing closeness centrality and identifies high ranked vertices. An important feature of our method is that the number of high ranked vertices returned can indicate whether the network indeed has high ranked vertices (e.g. Caida, Brightkite) or not (e.g. Gnutella). As part of our future research we plan to extend the algorithm to identify top-k vertices. This can be done by using other superimposed code strategies such as constructions using Reed-Solomon codes and randomly generated superimposed codes. We also plan to apply scalable group testing to other centrality metrics, such as betweenness centrality and eigen-vector centrality.

REFERENCES


