Efficient Maximum Closeness Centrality Group Identification

Chen Chen[1(✉)], Wei Wang[1], and Xiaoyang Wang[2]

[1] The University of New South Wales, Sydney, Australia
{cchen,weiw}@cse.unsw.edu.au
[2] University of Technology, Sydney, Australia
xiaoyang.wang@uts.edu.au

Abstract. As a key concept in the social networks, closeness centrality is widely adopted to measure the importance of a node. Many efficient algorithms are developed in the literature to find the top-k closeness centrality nodes. In most of the previous work, nodes are treated as irrelevant individuals for a top-k ranking. However, in many applications, it is required to find a set of nodes that is the most important as a *group*. In this paper, we extend the concept of closeness centrality to a set of nodes. We aim to find a set of k nodes that has the largest closeness centrality as a whole. We show that the problem is NP-hard, and prove that the objective function is monotonic and submodular. Therefore, the greedy algorithm can return a result with $1-1/e$ approximation ratio. In order to handle large graphs, we propose a baseline sampling algorithm (BSA). We further improve the sampling approach by considering the order of samples and reducing the marginal gain update cost, which leads to our order based sampling algorithm (OSA). Finally, extensive experiments on four real world social networks demonstrate the efficiency and effectiveness of the proposed methods.

Keywords: Closeness centrality · Random sampling · Top-k · Shortest path distance

1 Introduction

As a subject of broad and current interest, social networks have been widely studied for decades. A social network is usually represented as a graph $G = (V, E)$ where V denotes the set of nodes (users) and E denotes the set of edges (relationships among users). Centrality, which measures the importance of a node in a social network, has been a fundamental concept investigated in the social networks. There are different measurements of centrality developed for various purposes, such as closeness centrality [4], betweenness centrality [1], eigenvector centrality [5], etc. In this paper, we focus on the classic closeness centrality, which is defined as the inverse of the average distance from a node to all the other nodes in the social network. The distance between two nodes is calculated by the shortest path distance. The smaller the average distance of a node is, the

M.A. Cheema et al. (Eds.): ADC 2016, LNCS 9877, pp. 43–55, 2016.
DOI: 10.1007/978-3-319-46922-5_4

more important or more influential the node will be. To find the influential nodes (users) in a social network, many research efforts have been made to find the k nodes with the largest closeness centrality [10,13,14]. However, in many real applications, such as team formation, we may need to find a set of k users which has large closeness centrality as a *group*, instead of returning the k independent users in the top-k ranking. In this paper, we extend the definition of closeness centrality for a single node to a set of nodes. Specifically, the closeness centrality of a set S of nodes is defined as the inverse of the average distance from S to the nodes in G. And the distance from S to a node $u \in V$ is defined as the minimum distance from u to the nodes in S. The maximum closeness centrality group identification problem is to find a set of k nodes in the social network with the largest closeness centrality.

The challenges of the problem lie in two aspects. First, we show that the problem is NP-Hard, thus there is no polynomial time solution unless P = NP. Fortunately, we prove that the objective function is monotonic and submodular. It means we can obtain a result with $1 - 1/e$ approximation ratio by adopting a greedy framework. Second is that we still need the information of the all pairs shortest path distances even for the simple greedy algorithm, which is prohibitive to compute ($O(|V|^3)$ time) and store ($O(|V|^2)$ space) when the graph is large. There are some efficient index such as the network structure index techniques [15]. It partitions the graph into zones and approximates the shortest path distance between two nodes by using their distances to the same zone. However, this approximation offers no guarantee for the quality of the returned distance. In order to scale to large graphs, we propose a sampling based approach by extending the traditional sampling method for estimating the closeness centrality of a single node. In addition, we bring order into the samples, such that the nodes can be identified incrementally. Then we utilize the selected nodes to reduce the cost of computing the distances from the nodes to the samples. To further accelerate the process, we develop optimization techniques to reduce the updating cost for the less important nodes. Through experiments on real world social networks, we verify the efficiency and effectiveness of the proposed techniques.

The rest of the paper is organized as follows. Section 2 formally introduces the problem studied in this paper as well as the greedy algorithm based framework. Section 3 surveys related work. Section 4 presents the proposed sampling based algorithms. We demonstrate the efficiency and effectiveness of the proposed techniques on four real social networks in Sect. 5 and conclude the paper in Sect. 6.

2 Preliminary

We formally define the problem in this section. Then we analyze the properties of the objective function and introduce our greedy algorithm based framework.

2.1 Problem Definition

We consider a social network $G = (V, E)$ as an undirected connected graph, where V denotes the set of nodes, and E denotes the set of edges. $|V| = n$ and $|E| = m$. For nodes $u, v \in V$, the distance $d(u, v)$ between the two nodes is calculated as the shortest path distance and $d(u, u) = 0$. Then the classic closeness centrality of a node u is defined by the inverse average distance from u to the nodes in G.[1] By extending this definition, we can define the closeness centrality for a set of nodes as follows.

Definition 1 (Closeness Centrality for a Set of Nodes)**.** *Given a social network G and a set S ($S \subseteq V$) of nodes, the closeness centrality of S is denoted by $c(S)$, which is measured by the inverse average distance from S to all the nodes in G, i.e.,*

$$c(S) = \frac{n}{\sum_{v \in V} d(S, v)}$$

where $d(S, v) = \min\{d(u, v)\}$ for $u \in S$.

Problem Statement. Based on Definition 1, we define the *maximum closeness centrality group identification* (**MCGI**) problem as follows. Given a social network G and a positive integer k, the MCGI problem aims to find a set S^* of k nodes that has the largest closeness centrality, *i.e.,*

$$S^* = \arg\max_{S \subseteq V} \{c(S) \mid |S| = k\}. \tag{1}$$

The selected node set is call *seed set* and each node in the set is called a *seed*. According to Lemma 1, the MCGI problem is NP-Hard.

Lemma 1. *The maximum closeness centrality group identification problem is NP-Hard.*

Proof Sketch. The correctness of Lemma 1 can be proved by reducing from a known NP-hard problem "k-means clustering problem in the Euclidean space" [2]. The polynomial time reduction can be summarized as follows. Each object in the k-means clustering corresponds to a node in the social network. The k centers in the k-means clustering problem correspond to the selected k nodes. The Euclidean distance between objects corresponds to the shortest path distance between nodes in MCGI. Even if we can compute the shortest path distance in constant time, the hardness of the problem still remains the same.

[1] Note that there are different variants of the definition of closeness centrality, in this paper we focus on the classic closeness centrality.

2.2 Objective Function Analysis

According to Lemma 1, we know that there is no polynomial time solution for the MCGI problem. Fortunately, based on Lemma 2, the closeness centrality function for a set of nodes shown in Definition 1 has the following two properties.

- Monotonicity. Given any two sets $S,\ T \subseteq V$ with $S \subseteq T$, a function $f(x)$ is monotonic, if $f(S) \leq f(T)$.
- Submodularity. Given any two sets $S,\ T \subseteq V$ with $S \subseteq T$ and $v \in V \backslash T$, a function $f(x)$ is submodular, if $f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$.

Lemma 2. *The closeness centrality function for a set of nodes is monotonic and submodular.*

Proof Sketch. Assume there are two sets satisfying $S,\ T \subseteq V$ with $S \subseteq T$.

- Based on the definition of shortest path distance from a set of nodes to a single node, we have $d(S, u) \geq d(T, u)$ for $u \in V$, which means $c(S) \leq c(T)$. Thus the monotonic property is correct.
- Given two nodes u, v, where $v \in V \backslash T$ and $u \in V$, if $d(T, u) > d(T \cup \{v\}, u)$, we must have $d(T \cup \{v\}, u) = d(S \cup \{v\}, u) = d(v, u)$. Then we have $d(S \cup \{v\}, u) - d(S, u) \leq d(T \cup \{v\}, u) - d(T, u)$. If $d(T, u) = d(T \cup \{v\}, u)$, we have $d(S \cup \{v\}, u) - d(S, u) \leq d(T \cup \{v\}, u) - d(T, u) = 0$. Thus the submodular property holds based on the definition of closeness centrality. □

2.3 Greedy Algorithm Framework

Since the objective function is monotonic and submodular, we can utilize the greedy algorithm to iteratively select the node with the largest marginal gain (*i.e.,* the node will increase the closeness centrality of the set most by adding it). Through k iterations it can return a set S of k nodes with $1-1/e$ approximation ratio, *i.e.,* $c(S) \geq (1 - 1/e)c(S^*)$, where S^* is the optimal result. The details of the greedy algorithm framework are shown in Algorithm 1.

In Algorithm 1, S maintains the set of nodes already selected in the previous iterations. M is a matrix that stores the distance $d(S \cup \{u\}, v)$ $(v \in V)$ for each node u. *Score* is a vector that maintains the closeness centrality of adding u to S, *i.e.,* $c(S \cup \{u\})$ for each node $u \in V$. And $Score[u]$ equals $c(u)$ at the beginning. After initialization, we iteratively select the node with the largest marginal gain from $V \backslash S$ in Line 5. After selecting the node v with the largest marginal gain, we add it to S and update the distance matrix M as well as the *Score* value for each node $u \in V \backslash S$ from Line 7 to 10. The procedure is repeated until we find k nodes.

Analysis. The space complexity of Algorithm 1 is $O(n^2)$, since we need to store the all pairs distances in M. To compute the all pairs shortest paths, we can use the Floyd-Warshall algorithm which needs $O(n^3)$ time. Note that we can also use other algorithms to compute the all pairs shortest paths but it is not the major concern in this paper. In the node selection phase, we need $O(kn^2)$ time to select the k nodes as we need to update the distance matrix and the marginal gain for all the unselected nodes after each iteration.

Algorithm 1. Greedy Framework

```
Input  : G : a social network, k : a positive integer.
Output : S : a set of k nodes
1  S ← ∅ ;
2  M ← all pairs shortest path distances ;
3  Score ← {c(u) | u ∈ V};
4  while |S| < k do
5      v = arg max_{w∈V\S} Score[w] ;
6      S ← S ∪ {v} ;
7      foreach u ∈ V\S do
8          foreach w ∈ V do
9              if d(u, w) > d(v, w) then
10                 M[u, w] = d(v, w) ;
11         Score[u] ← c(S ∪ {u}) ;
12 return S;
```

3 Related Work

As a key concept in the social networks, centrality has been widely used to measure the importance of the nodes in a social network. Many centrality metrics [6,11,17] are proposed for different considerations to measure the importance of a node. For example, Google's PageRank is a variant of the eigenvector centrality. In this paper, we focus on the closeness centrality. The concept of closeness centrality is first considered in [3]. The definition of closeness centrality is formalized in [4] as the inverse of the average distance of a given node to all the other nodes in the graph. A major problem in calculating the closeness centrality is the scalability issue for large graphs. In [12,16], the authors use sampling approaches to find the top-1 node by utilizing the average distance to the sampled nodes. In [10], the authors extend the sampling approach for estimating the closeness centrality for all the nodes. Okamoto et al. [13] further extend the sampling framework to efficiently find the top-k nodes. To further improve the estimation performance, authors take advantage of a hybrid framework and weighted sampling techniques in [7–9]. [14] aims to find the top-k nodes exactly by leveraging the hierarchy structure of the graphs. In [18], the authors also study the group closeness centrality maximization problem. However, the techniques proposed are for disk-based graphs and only consider the case when the weight of edge is 1. While our method is memory-based and suitable for any weight setting.

4 Sampling Based Algorithms

Although the greedy approach in Algorithm 1 can return a result with a bounded approximation ratio, it suffers from serious limitations when scaling to large

graphs. As analyzed previously, it needs $O(n^2)$ space to store the all pairs distances, which is prohibitive for large graphs. If we do not store the all pairs distances in memory and only compute them on the fly when needed, the computation time is still not affordable. Moreover, we need considerable amount of time to update the marginal gain for each node in each iteration. Therefore, we propose two sampling based approaches to make it possible to scale to large graphs in this section.

4.1 Baseline Sampling Method

To make it possible for processing large graphs, a natural consideration is to utilize the sampling techniques. It is able to obtain a high quality result by accessing only a small part of the graph's information. In the previous works, sampling based approaches are developed to estimate the closeness centrality of a single node, or to identify the top-k closeness centrality nodes. The idea of estimating the closeness centrality of a single node u can be summarized as follows. We first randomly sample several nodes without replacement from V. Then we calculate the shortest path distances from u to all the samples. Next we use the average distance from u to all the samples as an estimation of the average distance of u to all the nodes. It is easy to see that the estimator is unbiased.

Motivated by the idea, we can use the average distance from a set S of nodes to all the samples as the estimation of the average distance of S to all the nodes. It can be formally expressed in Eq. (2).

$$\hat{c}(S) = \frac{l}{\sum_{v \in \mathcal{L}} d(S, v)}, \tag{2}$$

where $\mathcal{L}$ denotes the sample set of size l. It is easy to verify $1/\hat{c}(S)$ is an unbiased estimation of $1/c(S)$, *i.e.,* $1/c(S) = \mathbf{E}\left[1/\hat{c}(S)\right]$.

Based on the estimator, we can modify Algorithm 1 to apply the sampling technique and solve the MCGI problem. The idea is to use the estimated closeness centrality and marginal gain to replace the exact calculated value in the greedy framework. To be more specific, we first sample l nodes from V. For the distance matrix M, it stores the distances from each node $u \in V$ to all the samples, *i.e.,* $d(S \cup \{u\}, v)$ $(v \in \mathcal{L})$. Initially, it only stores the distances from all the nodes to the l samples. This can be done by running a single source shortest path algorithm from every sample. Similarly, $Score[u]$ stores $\hat{c}(S \cup \{u\})$. Then we iteratively select k nodes with the largest marginal gain based on the sampling method. The pseudo-code is omitted due to the space limitation.

Analysis. The space cost is $O(nl)$ for the baseline sampling algorithm, since only the distances from all the nodes to the l samples are maintained. In order to compute the initial matrix M, we can run l times single source shortest path algorithm (*e.g.,* Dijkstra algorithm) which requires $O(l(m + n \log n))$ time. We also need $O(kln)$ time to select the k nodes. Due to the sampling techniques, the result will have $1 - 1/e - \epsilon$ approximation ratio, where ϵ is the error introduced

by sampling. From previous studies [10,12,16] we know the estimation procedure converges quickly with the sample size l.

4.2 Order Based Sampling Method

Although the baseline sampling algorithm can greatly reduce the space cost and time complexity, there is still room for improvement.

- The first limitation is that it strictly separates the shortest path distance calculation phase and the node selection phase. As introduced later, we can further reduce the cost if incrementally doing the sampling and the seed selection.
- The second limitation is that after selecting a node, it needs to update the marginal gain for all the nodes, which is costly when n is large.

Based on the observations, we present two optimized techniques to further reduce the cost of baseline sampling algorithm.

Incremental Sampling and Node Selection. We first explain the motivation. Suppose we already select certain nodes by using $l' < l$ samples, then we can use these selected nodes (seed set) to reduce the calculation cost of shortest path distances for the rest $l - l'$ samples. Following is a motivating example.

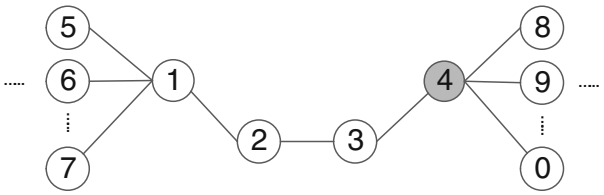


Fig. 1. Motivating example of incremental sampling and node selection

Example 1. As shown in Fig. 1, assume $k = 2$ and the weight of each edge is 1. We use half of the samples to select the first seed v_4 hence currently $S = \{v_4\}$. Then we examine the rest samples to find the second seed. Suppose v_3 is the next sample, then we start running a Dijkstra's algorithm from v_3. The property of Dijkstra's algorithm is that it will incrementally reach the nodes close to the source node. So in the figure, v_3 first reaches nodes v_2 and v_4 with shortest path distance of 1. Remember that v_4 is the node already selected. Then we do not need to further compute the distance from all the other nodes to v_3, as their distances are greater than that of v_4. The reason is that adding any of them to S will not change the distance from S to the sample v_3. Consequently, their distances to v_3 are all recorded as $1 = d(v_3, v_4)$, that is for $v' \in V \backslash \{v_2, v_3, v_4\}$, it holds $d(S \cup \{v'\}, v_3) = 1$.

According to the motivating example, if we can incrementally do the sampling and node selection, we will significantly reduce the cost of computing shortest path distances from nodes to the samples. To fulfill it, we firstly divide the sample into k partitions $\{P_1, P_2, ..., P_k\}$. For the i-th node selection, we compute the distances from all the nodes to the samples in P_i by considering the selected $i-1$ nodes. Then we can obtain the node with the largest marginal gain by considering the estimation due to samples in $\cup_{j=1}^{i} P_j$. The details of the algorithm are shown in Algorithm 2.

Algorithm 2. Order Based Sampling

Input : G : a social network, k : a positive integer, l : sample size.
Output : S : a set of k nodes

1 $S \leftarrow \emptyset$; $Score \leftarrow \emptyset$; $M \leftarrow \emptyset$;
2 Get samples and partitions $\{P_1, P_2, ..., P_k\}$; `/* partition of l samples */`;
3 **for** i from 1 to k **do**
4 Compute the distance from all nodes to P_i with S as constraint ;
5 Update $Score$ for $u \in V \backslash S$ based on the samples in $\cup_{j=1}^{i} P_j$;
6 $v = \arg\max_{w \in V \backslash S} Score[w]$;
7 $S \leftarrow S \cup \{v\}$;
8 **foreach** $u \in V \backslash S$ **do**
9 **foreach** $w \in \cup_{j=1}^{i} P_j$ **do**
10 **if** $d(u, w) > d(v, w)$ **then**
11 $M[u, w] = d(v, w)$;
12 **return** S;

In Algorithm 2, we sample l nodes and divide them into k partitions in Line 2. To get l samples without replacement, we can do a random permutation on all the nodes and select the first l nodes. In such case, each sample has a rank induced by the permutation. Then each partition can store the samples based on the order, *i.e.,* P_i stores the $(i-1)l/k$-th to il/k-th sampled nodes.

In the i-th iteration, we compute the distances from all the nodes to samples in P_i with S as the constraint. It means when running Dijkstra's algorithm from a sample v', we can early stop if we get the shortest path distance from v' to a seed node u' in S. In Line 5, we update the $Score$ value for nodes based on the samples in $\cup_{j=1}^{i} P_j$. Then we select the node v with the largest marginal gain and add it to S. Finally, we update the distance matrix, *i.e.,* $d(S \cup \{u\}, w)$ for $u \in V \backslash S$ and $w \in \cup_{j=1}^{i} P_j$. The algorithm terminates after k iterations.

Correctness of Algorithm 2. The aim of the algorithm is to select the node with the largest marginal gain from $V \backslash S$ in each iteration. The input of the iteration is the seed set S from previous $i-1$ iterations. Since the proposed optimization only prunes the unnecessary calculation, the distance matrix M and $Score$ vector are exactly calculated based on the samples in $\cup_{j=1}^{i} P_j$ given S. Moreover, $\cup_{j=1}^{i} P_j$

consists of the first $|\cup_{j=1}^{i} P_j|$ samples of the l samples, as we partition the samples based on the sample order (*i.e.,* permutation order). Consequently, the case in iteration i amounts to selecting the node with the largest marginal gain, based on the already selected $i-1$ nodes with $|\cup_{j=1}^{i} P_j|$ samples. Therefore the algorithm is guaranteed to be correct. Although the incremental selection strategy may affect the accuracy of estimation, the quality drop is negligible as shown in the experimental evalutions.

Update Optimization. In the baseline sampling algorithm, after selecting one node in an iteration, we need to update the *Score* vector for all the other nodes. However, the closeness centrality tends to be very small for most of the nodes in a social network. Usually we have $k << n$. As a result, the nodes with small centralities will never be selected into the results. If we can avoid updating their marginal gains we will save a large amount of computational cost. Following is a motivating example.

Example 2. Suppose node v_1 is a less important node with $c(v_1) = 0.0001$. In the i-th iteration, we find a node v_2 with marginal gain of 0.05. Due to the submodular property of the objective function, the marginal gain of v_1 must be smaller than $c(v_1) < 0.05$. Thus we can safely prune v_1 from the i-th node selection without updating its marginal gain.

Based on the motivating example, we come up with the *update when needed strategy.* The idea is that in the i-th iteration, we do not calculate the marginal gain for node $v \in V$ unless necessary. It is easy to adopt this strategy by modifying Algorithm 2. For each node $u \in V \backslash S$, $Score[u]$ stores its centrality or values calculated in the previous iteration, so does for the distance matrix M. In the i-th iteration, we first calculate the distance of nodes to the samples in P_i, then update the matrix and the score vector based on P_i. Next we sort the nodes decreasingly based on their values in *Score*. We continuously pop nodes from the queue and calculate their true marginal gains, until the largest marginal gain found is greater than the top value of the queue. Then we can safely prune all the untouched nodes from this iteration and select the node with the largest marginal gain as the next seed. Through this way, we can reduce the update cost for many unpromising nodes, which will greatly improve the efficiency when n is large.

5 Experiment

In this section, we present the results of a comprehensive performance study to evaluate the efficiency and effectiveness of the proposed techniques in this paper.

5.1 Experiment Setup

In the experiments, we report the response time of finding k nodes to evaluate the efficiency of the algorithms. We also report the closeness centrality of the returned node set to measure the effectiveness.

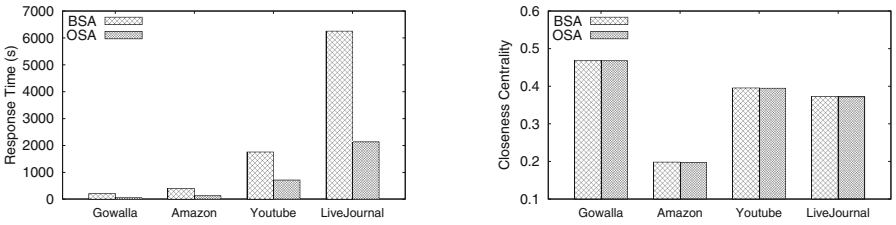
Algorithms and Workload. We evaluate the performance of two proposed algorithms, baseline sampling algorithm (**BSA**) and order based sampling algorithm (**OSA**). We omit the greedy algorithm in Algorithm 1, since it needs to compute and store all pairs shortest path distances. Even for a small graph with 50,000 nodes, it will need more than one day to return the results. We set the sample size for both BSA and OSA as 1000 to make a tradeoff between the accuracy and efficiency. In the experiments, we vary k from 1 to 50 with 50 by default. For each algorithm, we run 20 times and report the average performance.

Datasets. To demonstrate the effectiveness and efficiency of our methods, we conduct experiments on 4 real world social networks. The parameters of the datasets [2] are reported in Table 1. The diameter is defined as the longest shortest path distance in the graph.

Table 1. The summary of datasets

Dataset	n	m	Diameter
Gowalla	196,591	950,327	14
Amazon	334,863	925,872	44
Youtube	1,134,890	2,987,624	20
LiveJournal	3,997,962	34,681,189	17

Implementation Environment. All experiments are carried out on a PC with Intel Xeon 2.30 GHz and 96 G RAM. The operating system is Redhat. All algorithms are implemented in C++ and compiled with GCC 4.8.2 with -O3 flag.



(a) Efficiency (b) Effectiveness

Fig. 2. Performance evaluation on all the datasets

[2] https://snap.stanford.edu/data/.

5.2 Efficiency Evaluation

In this section we evaluate the efficiency of the algorithms through response time. In the first set of experiments, Fig. 2a reports the response time of BSA and OSA on all the datasets under the default settings. As the increase of dataset size, the response time grows for both algorithms. Because the cost of computing the distance from nodes to the samples and calculating the marginal gain will increase with the growing graph size. OSA constantly outperforms BSA by up to 4 times acceleration, due to the pruning in computing shortest path distances and updating the marginal gains.

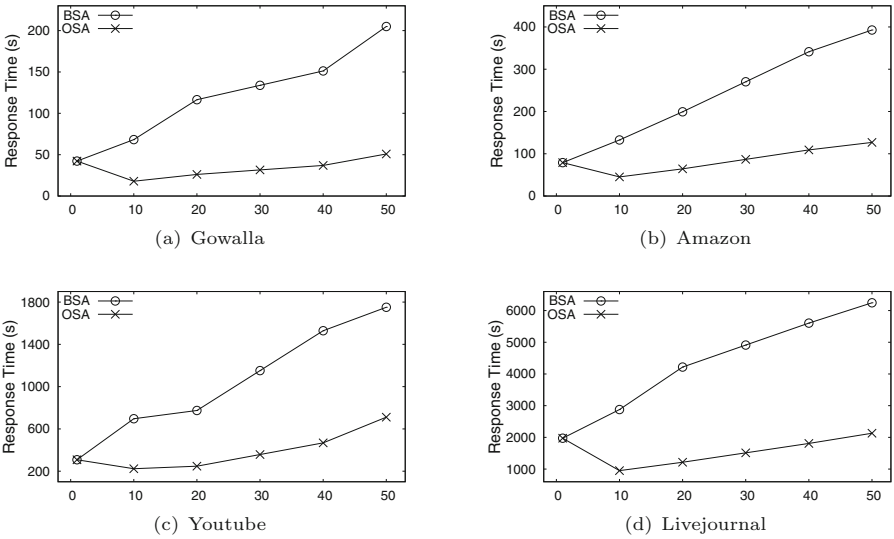


Fig. 3. Efficiency evaluation by varying k

In Fig. 3, we report the response time by varying k from 1 to 50 on four datasets. When k equals 1, the response time of both algorithms is the same. It is because when k equals 1, there is no incremental node selection optimization and the updating optimization for OSA. Under the same sample size and $k = 1$, the procedure of OSA is the same as that of BSA. When k increases, the response time of BSA increases because of the increase of cost in the node selection. For OSA, the response time firstly drops then increases, because when k is larger than 1, OSA can take advantage of incremental node selection to reduce the cost of calculating shortest path distances to the samples. However, when k becomes larger, the node selection cost and marginal gain updating cost increase, which leads to the increase of the running time.

5.3 Effectiveness Evaluation

In this section, we evaluate the effectiveness of the proposed algorithms. In Fig. 2b, we report the closeness centrality on all the datasets under default settings. As can be seen, the quality of the results returned by OSA is almost the same as that of BSA. Only in few cases there is a very slight drop in the closeness centrality returned by OSA. Since for each node selection in OSA, it only utilizes part of the samples for estimating, *i.e.,* $\cup_{j=1}^{i} P_j$ for selecting the i-th node. While BSA always uses the full samples to do the estimation. Also note that the closeness centrality of returned nodes is decided by the diameter of the graph. For graphs with small diameters, it tends to return a set of nodes with large closeness centrality when k is identical. In Fig. 4, we report the closeness centrality by varying k from 1 to 50. When k increases, the closeness centrality increases for the returned node set. The quality difference of the results by both algorithms is very small. For less dense graph with large diameters such as the Amazon dataset, the closeness centrality increases slowly when k is small (1 to 10).

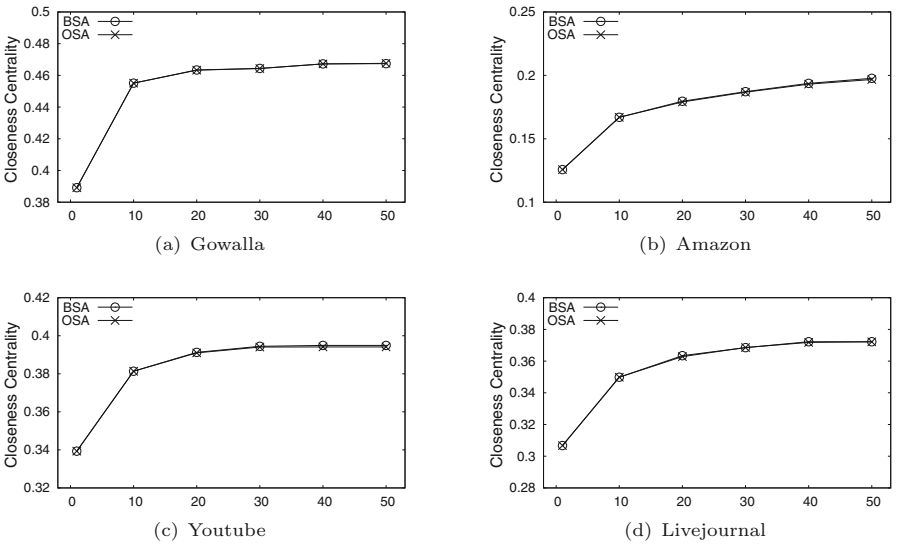


Fig. 4. Effectiveness evaluation by varying k

6 Conclusion

In this paper, we consider closeness centrality for a set of nodes and aim to find the set with the largest closeness centrality. We show the problem is NP-Hard. By proving the monotonic and submodular properties of the objective function, we present a greedy framework which can achieve $1 - 1/e$ approximation ratio. Unfortunately, naïve implementation of the greedy framework will result

in large space and time cost. To be able to scale to large graphs, we present two sampling based algorithms, BSA and OSA, respectively. OSA significantly accelerates BSA due to the optimizations in the shortest path distance computation and the updating procedure. By conducting extensive experiments on four real social networks, we demonstrate the efficiency and effectiveness of the proposed techniques.

References

1. Abboud, A., Grandoni, F., Williams, V.V.: Subcubic equivalences between graph centrality problems, APSP and diameter. In: SODA, pp. 1681–1697 (2015)
2. Aloise, D., Deshpande, A., Hansen, P., Popat, P.: NP-hardness of euclidean sum-of-squares clustering. Mach. Learn. **75**(2), 245–248 (2009)
3. Bavelas, A.: A mathematical model for small group structures. Hum. Organ. (1948)
4. Bavelas, A.: Communication patterns in task oriented groups. J. Acoust. Soc. Am. **22**(6), 726–730 (1950)
5. Bonacich, P., Lloyd, P.: Eigenvector centrality and structural zeroes and ones: when is a neighbor not a neighbor? Soc. Netw. **43**, 86–90 (2015)
6. Brandes, U.: A faster algorithm for betweenness centrality. J. Math. Sociol. **25**, 163–177 (2001)
7. Chechik, S., Cohen, E., Kaplan, H.: Average distance queries through weighted samples in graphs and metric spaces: high scalability with tight statistical guarantees. In: APPROX/RANDOM (2015)
8. Cohen, E.: All-distances sketches, revisited: HIP estimators for massive graphs analysis. In: PODS (2014)
9. Cohen, E., Delling, D., Pajor, T., Werneck, R.F.: Computing classic closeness centrality, at scale. In: COSN (2014)
10. Eppstein, D., Wang, J.: Fast approximation of centrality. In: SODA (2001)
11. Freeman, L.C.: Centrality in social networks conceptual clarification. Soc. Netw. **1**, 215–239 (1978)
12. Indyk, P.: Sublinear time algorithms for metric space problems. In: STOC (1999)
13. Okamoto, K., Chen, W., Li, X.-Y.: Ranking of closeness centrality for large-scale social networks. In: Preparata, F.P., Wu, X., Yin, J. (eds.) FAW 2008. LNCS, vol. 5059, pp. 186–195. Springer, Heidelberg (2008)
14. Olsen, P.W., Labouseur, A.G., Hwang, J.: Efficient top-k closeness centrality search. In: ICDE (2014)
15. Rattigan, M.J., Maier, M., Jensen, D.: Graph clustering with network structure indices. In: ICML, pp. 783–790 (2007)
16. Thorup, M.: Quick k-Median, k-Center, and facility location for sparse graphs. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, p. 249. Springer, Heidelberg (2001)
17. Wasserman, S., Faust, K.: Social network analysis: methods and applications. Cambridge University Press, Cambridge (1994)
18. Zhao, J., Lui, J.C., Towsley, D., Guan, X.: Measuring and maximizing group closeness centrality over disk-resident graphs. In: WWW 2014 Companion, pp. 689–694 (2014)