Efficient Centrality Monitoring for Time-Evolving Graphs

Yasuhiro Fujiwara[1], Makoto Onizuka[1], and Masaru Kitsuregawa[2]

[1] NTT Cyber Space Laboratories, Japan
{fujiwara.yasuhiro,onizuka.makoto}@lab.ntt.co.jp
[2] The University of Tokyo, Japan
kitsure@tkl.iis.u-tokyo.ac.jp

Abstract. The goal of this work is to identify the nodes that have the smallest sum of distances to other nodes (the lowest closeness centrality nodes) in graphs that evolve over time. Previous approaches to this problem find the lowest centrality nodes efficiently at the expense of exactness. The main motivation of this paper is to answer, in the affirmative, the question, 'Is it possible to improve the search time without sacrificing the exactness?'. Our solution is Sniper, a fast search method for time-evolving graphs. Sniper is based on two ideas: (1) It computes approximate centrality by reducing the original graph size while guaranteeing the answer exactness, and (2) It terminates unnecessary distance computations early when pruning unlikely nodes. The experimental results show that Sniper can find the lowest centrality nodes significantly faster than the previous approaches while it guarantees answer exactness.

Keywords: Centrality, Graph mining, Time-evolving.

1 Introduction

In graph theory, the facility location problem is quite important since it involves finding good locations for one or more facilities in a given environment. Solving this problem starts by finding the nodes whose distances to other nodes is the shortest in the graph, since the cost it takes to reach all other nodes from these nodes is expected to be low. In graph analysis, the centralities based on this concept are **closeness**. In this paper, the closeness centrality of node u, C_u, is defined as the sum of distances from the node to other nodes.

The naive approach, the exact computation of centrality, is impractical; it needs distances of all node pairs. This led to the introduction of approximate approaches, such as the annotation approach [13] and the embedding approach [12,11], to estimate centralities. These approaches have the advantage of speed at the expense of exactness. However, approximate algorithms are not adopted by many practitioners. This is because the optimality of the solution is not guaranteed; it is hard for approximate algorithms to identify the lowest centrality node exactly. Furthermore, the focus of traditional graph theory has been limited to just 'static' graphs; the implicit assumption is that nodes and edges never change. Recent years have witnessed a dramatic increase in the availability of graph datasets that comprise many thousands and sometimes even millions of time-evolving nodes; a consequence of the widespread availability of electronic databases and the Internet. Recent studies on large-scale graphs are discovering several important principles of time-evolving graphs [10,8]. Thus demands for the efficient analysis of time-evolving graphs are increasing. We address the following problem in this paper:

Given: *graph* $G[t] = (V[t], E[t])$ *at time* t *where* $V[t]$ *is a set of nodes and* $E[t]$ *is a set of edges at time* t.
Find: *the nodes that have the lowest closeness centrality in graph* $G[t]$.

We propose a novel method called Sniper that can efficiently identify the lowest centrality nodes in time-evolving graphs. To the best of our knowledge, our approach is the first solution to achieve both exactness and efficiency at the same time in identifying the lowest centrality nodes from time-evolving graphs.

1.1 Problem Motivation

The problem tackled in this paper must be overcome to develop the following important applications.

Networks of interaction have been studied for a long time by social science researchers, where nodes correspond to people or organizations, and edges represent some type of social interaction. The question of 'which is the most important node in a network?' is being avidly pursued by scientific researchers. An important example is the network obtained by considering scientific publications. Nodes in this case are researchers, papers, books, or entire journals, and edges correspond to co-authorship or citations. This kind of network generally grows very rapidly over time. For example, the collaboration network of scientists in the database area contains several tens of thousands of authors and its rate of growth is increasing year by year; there are several thousand new authors each year [5]. The systematic construction of such networks was introduced by Garfield, who later proposed a measure of standing for journals that is still in use. This measure, called impact factor, is defined as the number of citations per published item [6]. Basically, the impact factor is a very simple measure, since it corresponds to the degree of the citation network. However the degree is a local measure, because the value is only determined by the number of adjacent nodes. That is, if a high-degree node lies in an isolated community of the network, the influence of the node is very limited.

Closeness centrality is a global centrality measure since it is computed by summing the distances to all other nodes in a graph. Therefore, it is an effective measure of influence on other nodes. The most influential node can be effectively detected as the lowest closeness centrality node by monitoring time-evolving graphs. Nascimento et al. analyzed SIGMOD's co-authorship graph [9] They successfully discovered that L. A. Rowe, M. Stonebraker, and M. J. Carey were the most influential researchers from 1986 to 1988, 1989 to 1992, and 1993 to 2002, respectively. All these three are very famous and important researchers in the database community.

The remainder of this paper is organized as follows. Section 2 describes related work. Section 3 overviews some of the background of this work. Section 4 introduces the main ideas of Sniper. Section 5 discusses some of the topics related to Sniper. Section 6 gives theoretical analyses of Sniper. Section 7 reviews the results of our experiments. Section 8 provides our brief conclusion.

2 Related Work

Many papers have been published on approximations of node-to-node distances. The previous distance approximation schemes are distinguished into two types: annotation types and embedding types. Rattigna et al. studied two annotation schemes [13]. They randomly select nodes in a graph and divide the graph into regions that are connected, mutually exclusive, and collectively exhaustive. They give a set of annotations to every node from the regions. Distances are computed by the annotations. They demonstrated their method can compute node distances more accurately than the embedding approaches. However, this method can require $O(n^2)$ space and $O(n^3)$ time to estimate the lowest centrality nodes as described in their paper.

The Landmark technique is an embedding approach [7,12], and estimates node-to-node distance from selected nodes at $O(n)$ time. The minimum distance via a landmark node is utilized as node distance in this method. Another embedding technique is Global Network Positioning, which was studied by Ng et al. [11]. Node distances are estimated from the L_p norm between node pairs. These embedding techniques require $O(n^2)$ space since all n nodes hold distances to $O(n)$ selected nodes. Moreover, they require $O(n^3)$ time to identify the lowest centrality node. This is because they take $O(n)$ time to estimate a node pair distance and need the distances of n^2 node pairs to compute centralities of all nodes.

3 Preliminary

In this section, we introduce the background to this paper. Social networks and others can be described as graph $G = (V, E)$, where V is the set of nodes, and E is the set of edges. We use n and m to denote the number of nodes and edges, respectively. That is $n = |V|$ and $m = |E|$. A path from node u to v is the sequence of nodes linked by edges, beginning with node u and ending at node v. A path from node u to v is the shortest path if and only if the number of nodes in the path is the smallest possible among all paths from node u to v. The distance between node u and v, $d(u, v)$, is the number of edges in the shortest path connecting them in the graph. Therefore $d(u, u) = 0$ for every $u \in V$, and $d(u, v) = d(v, u)$ for $u, v \in V$. The closeness centrality of node u, C_u, is the sum of the distances from the node to any other node, and computed as $\sum_{v \in V} d(u, v)$.

4 Centrality Monitoring

In this section, we explain the two main ideas underlying Sniper. The main advantage of Sniper is to exactly and efficiently identify the lowest closeness centrality nodes in time-evolving graphs. While we focus on undirected and unweighted graphs in this section, our approach can be applied to weighted or directed graphs as described in Section 5.1. Moreover, we can handle range queries (find the nodes whose centralities are less than a given threshold) and K-best queries (find the K lowest centrality nodes) as described in Section 5.2. For ease of explanation, we assume that no two nodes will have exactly the same centrality value and that one node is added to a time-evolving graph at each time tick. These assumptions can be eliminated easily. And all proofs in this section are omitted due to the space limitations.

4.1 Ideas Behind Sniper

Our solution is based on the two ideas described below.

Node aggregation. We introduce approximations to reduce the high cost of the existing approaches. Instead of computing the exact centrality of every node, we approximate the centrality, and use the result to efficiently prune high-centrality nodes.

For a given graph with n nodes and m edges, we create an approximate graph of n' nodes and m' edges ($n' < n, m' < m$) by aggregating 'similar' nodes in the graph. For the approximate graph, $O(n' + m')$ time is required for Sniper to compute the approximate centralities, while the existing approximate algorithm requires $O(n^2)$ time as described in Section 2. We exploit the Jaccard coefficient to find similar nodes, and then aggregate the original nodes to create node groups. We refer to such groupings as **aggregate** nodes.

This new idea has the following two major advantages. First, we can find the answer node exactly; the node that has the lowest centrality is never missed by this approach. This is because our approximate graphs guarantee the lower bounding distances. This means that we can safely discard unpromising nodes at low CPU cost. The second advantage is that this idea can reduce the number of nodes that must be processed to compute centralities, as well as reducing the computation cost for each node. That is, we can identify the lowest centrality node among a large number of nodes efficiently.

Tree estimation. Although our approximation technique is able to discard most of the unlikely nodes, we still rely on exact centrality computation to guarantee the correctness of the search results. Here we focus on reducing the cost of this computation.

To compute the exact centrality of a node, distances to all other nodes from the node have to be computed by breadth-first search (BFS). But clearly the exhaustive exploration of nodes in a graph is not computationally feasible, especially for large graphs. Our proposal exploits the following idea: If a node cannot be the lowest centrality node, we terminate subsequent distance computations as being unnecessary.

Our search algorithm first holds a candidate node, which is expected to have low centrality. We then estimate the distances of unexplored nodes in the distance computation from a single BFS-tree to obtain the lower centrality bound. In the search process, if the lower centrality bound of a node gives a value larger than the exact centrality of the candidate node, the node cannot be the lowest centrality node in the original graph. Accordingly, unnecessary distance computations can be terminated early.

4.2 Node Aggregation

Our first idea involves aggregating nodes of the original graph, which enables us to compute the lower centrality bound and thus realize reliable node pruning.

Graph Approximation. We reduce the original graph size in order to compute approximate centralities at low computation cost. To realize efficient search, given original graph G with n nodes and m edges, we compute n' nodes and m' edges in the approximate graph G'. That is, the original graph $G = (V, E)$ is collapsed to yield the approximate graph $G' = (V', E')$. We first describe how to compute the edges of the approximate graph, and then show our approach of original node aggregation.

For the aggregate nodes u' and v', there is an edge, $\{u', v'\} \in E'$, if and only if there is at least one edge between aggregated original nodes in u' and v'. This definition is important in computing the lower centrality bound. Formally, we obtain the edges between aggregate node u' and v' as follows:

Definition 1 (Node aggregation). *In the approximate graph G', node u' and v' have an edge if and only if:*

$$(1) u' \neq v', \quad (2) \exists \{u, v\} \;\; s.t. \;\; u \in u' \cap v \in v' \tag{1}$$

where $u \in u'$ indicates that aggregate node u' contains original node u.

To reduce the approximation error, we aggregate similar nodes. As described above, the aggregate nodes share an edge if and only if there is at least one edge between the original nodes that have been aggregated. Therefore, the approximation error decreases as the number of neighbors shared by the aggregated nodes increases. For this reason, we utilize the Jaccard coefficient since it is a simple and natural measure of similarity between sets [4].

Let N_u and N_v be sets of neighbors (adjacent nodes) of nodes u and v, respectively; the Jaccard coefficient is defined as $|N_u \cap N_v|/|N_u \cup N_v|$, i.e. the size of the intersection of the sets divided by the size of their union. We aggregate node u and v if the most similar node of u is node v, this yields good approximations. Note, we do not aggregate nodes u and v if the size of their intersection is less than one half the size of their union to avoid aggregating dissimilar nodes.

If one node is added to a time-evolving graph, we compute its most similar node to update the approximate graph. The naive approach to compute the most similar node for the added node is to compute the similarities for all nodes. We, on the other hand, utilize the following lemma to efficiently update the most similar node:

Lemma 1 (Update the most similar nodes). *For the added node, the most similar node is at most two hops apart.*

By using the above lemma, we first obtain the nodes which are one and two hop away from the added node in the search process. And we compute similarity for the added node and update the most similar node. And we link the aggregate nodes with Definition 1.

Even though we assume a single node is added for time-evolving graphs in each time tick, Lemma 1 can also be applied for the case of single node deletion. That is we can efficiently update the most similar node with Lemma 1 for node deletion. We iterate the above procedure for each node if several nodes are added. If one edge is added/deleted, we delete one connected node and add the node.

Lower Bounding Centrality. Given an approximate graph, we compute the approximate centrality of node u' as follows:

Definition 2 (Approximate closeness centrality). *For the approximate graph, the approximate closeness centrality of node u', $C_{u'}$, is computed as follows:*

$$C_{u'} = \sum_{v' \in V'} \{d(u', v') \cdot |v'|\} \tag{2}$$

where $d(u', v')$ is node distance in the approximation graph (i.e. the number of hops from node u' to v') and $|v'|$ is the number of original nodes aggregated within node v'.

We can provide the following lemma about the centrality approximation:

Lemma 2 (Approximate closeness centrality). *For any node in the approximate graph, $C_{u'} \leq C_u$ holds.*

Lemma 2 provides Sniper with the property of finding the exact answer as is described in Section 6.

4.3 Tree Estimation

We introduce an algorithm for computing original centralities efficiently. We terminate subsequent distance computations from a node if the estimate centrality of the node is larger than the exact centrality of the candidate node. In this approach, we compute lower bounding distances of unexplored nodes via BFS to estimate the lower centrality bound of a node. Estimations are obtained from a single BFS-tree.

Notation. We first give some notations for the estimation. In the search process, we construct the BFS-tree rooted at a selected node. As a result, the selected node forms layer 0. The direct neighbors of the node form layer 1. All nodes that are i hops apart from the selected node form layer i. We later describe our approach to selecting the node.

Next, we check by BFS that the exact centralities of other nodes in the tree are lower than the exact centrality of the candidate node. We define the set of nodes explored by BFS as V_{ex}, and the set of unexplored nodes as $V_{un}(= V - V_{ex})$. $d_{max}(u)$ is the maximum distance of the explored node from node u, that is $d_{max}(u) = \max\{d(u,v) : v \in V_{ex}\}$. Moreover, we define the explored layers of the tree as L_{ex} if and only if there exists at least one explored node in the layer. Similarly we define the unexplored layers as L_{un} if and only if there exists no explored node in the layer. The layer number of node u is denoted as $l(u)$.

Centrality Estimation. We define how to estimate the centrality of a node. We estimate the closeness centrality of node u via BFS as follows:

Definition 3 (Estimate closeness centrality). *For the original graph, we define the following centrality estimate of node u, $\hat{C}_u$, to terminate distance computation in BFS:*

$$\hat{C}_u = \sum_{v \in V_{ex}} d(u,v) + \sum_{v \in V_{un}} e(u,v) \tag{3}$$

$$e(u,v) = \begin{cases} d_{max}(u) & (v \in V_{un} \cap L_{ex}) \\ d_{max}(u) + \min\{|l(v) - l(w)|\} - 1 & (v \in L_{un}, w \in L_{ex}) \end{cases}$$

The estimation is the same as exact centrality if all nodes are explored (i.e. $V_{ex} = V$) in Equation (3). To show the property of the centrality estimate, we introduce the following lemma:

Lemma 3 (Estimate closeness centrality). *For the original graph, $\hat{C}_u \leq C_u$ holds in BFS.*

This property enables Sniper to identify the lowest centrality node exactly.

The selection of the root node of the tree is important for efficient pruning. We select the lowest centrality node of the previous time tick as the root node. There are two reasons for this approach. The first is that this node and nearby nodes are expected to have the lowest centrality value, and thus are likely to be the answer node after node addition. In the case of time-evolving graphs, small numbers of nodes are continually being added to the large number of already existing nodes. Therefore, there is little difference between the graphs before and after node addition. In addition, we can more accurately estimate the centrality value of a node if the node is close to the root node; this is the second reason. This is because our estimation scheme is based on distances from the root node.

Algorithm 1. Sniper

```
Input: $G[t] = (V, E)$, a time-evolving graph at time $t$
       $u_{add}$, the node added at time $t$
       $u_{low}[t-1]$, the previous lowest centrality node
Output: $u_{low}[t]$: the lowest centrality node.
 1: //Update the approximate graph
 2: update the approximate graph by the update algorithm;
 3: //Search for the lowest centrality node
 4: $V_{exact} \leftarrow$ empty set;
 5: compute the BFS-tree of node $u_{low}[t-1]$;
 6: compute $\theta$, the exact centrality of node $u_{low}[t-1]$;
 7: for each node $v' \in V'$ do
 8:    compute $C_{v'}$ in the approximate graph by the estimation algorithm;
 9:    if $C_{v'} \leq \theta$ then
10:       for each node $v \in v'$ do
11:          append node $v \rightarrow V_{exact}$;
12:       end for
13:    end if
14: end for
15: for each node $v \in V_{exact}$ do
16:    compute $C_v$ in the original graph by the estimation algorithm;
17:    if $C_v < \theta$ then
18:       $\theta \leftarrow C_v$;
19:       $u_{low}[t] \leftarrow v$;
20:    end if
21: end for
22: return $u_{low}[t]$;
```

4.4 Search Algorithm

Our main approach to finding the lowest centrality node is to prune unlikely nodes by using our approximation, and then confirm by exact centrality computations whether the viable nodes are the answer. However, an important question is which node should be selected as the candidate in time-evolving graphs. We select the previous lowest centrality node as the candidate. This node likely to have lowest centrality as described in Section 4.3. After we construct the BFS-tree, the exact centrality of the candidate node can be directly obtained with this approach.

Algorithm 1 shows the search algorithm that targets the lowest closeness centrality node. In this algorithm, $u_{low}[t]$, $u_{low}[t-1]$ and u_{add} indicate the lowest centrality node, the previous lowest centrality node, and the added node, respectively. V_{exact} represents the set of nodes for which we compute exact centralities.

The algorithm can be divided into two phases: update and search. In the update phase, Sniper computes the approximate graph by the update algorithm (line 2). In the search phase, Sniper first computes the BFS-tree of the answer node of the last time tick (line 5) and θ (line 6). If the approximate centrality of a node is larger than θ, we prune the node since it cannot be the lowest centrality node. Otherwise, Sniper appends aggregated original nodes to V_{exact} (lines 9-13), and then computes exact centralities to identify the lowest centrality node (lines 15-21).

5 Extension

In this section, we give a discussion of possible extensions to Sniper.

5.1 Directed or Weighted Graphs

We focus on undirected and unweighted graphs in this paper, but Sniper can also handle directed or weighted graphs effectively. As described in Section 4.2, approximate graphs have an edge if and only if there is at least one edge between aggregated nodes in an undirected and unweighted graph. However, we must modify how approximate graphs are constructed if we are to handle other kinds of graphs.

For directed graphs, we apply Definition 1 for each direction to handle directed edges of approximate graphs. For weighted graphs, we choose the lowest value of the weights of the original edges as the weight of the aggregated edge to compute the lower bound of exact centralities.

To estimate centrality values for weighted graphs, we can directly apply Definition 2. For weighted graphs, however, we need a little modification. We estimate distance from node u to v as $d_{max}(u) + \min\{\omega(v, w) : w \in V \setminus v\}$ where $\omega(v, w)$ is the weight of edge $\{v, w\}$.

5.2 Other Types of Queries

Although the search algorithm described here identifies the node that has the lowest centrality, the proposed approach can be applied to range queries and K-best queries. Range queries find the nodes whose centralities are less than a given threshold, while K-best queries find the K lowest centrality nodes.

For range queries, we utilize the given search threshold as θ, instead of the exact centrality of the previous time tick (i.e., we do not use the candidate). We compute approximate centralities of all nodes and prune unlikely nodes using the given θ; we confirm the answer nodes by calculating exact centralities.

For K-best queries, we first compute the exact centralities at time t of all K answer nodes in the last time tick. Next, we select the K-th lowest exact centrality as θ. Subsequent procedures are the same as for the case of identifying the lowest centrality node.

6 Theoretical Analysis

This section provides theoretical analyses that confirm the accuracy and complexity of Sniper. Let n be the number of nodes and m the number of edges.

We prove that Sniper finds the lowest centrality node accurately (without fail) as follows:

Theorem 1 (Find the lowest centrality node). *Sniper guarantees the exact answer when identifying the node whose centrality is the lowest.*

Proof. Let u_{low} be the lowest centrality node in the original graph, and θ_{low} be the exact centrality of u_{low} (i.e., θ_{low} is the lowest centrality). Also let θ be the candidate centrality in the search process.

In the approximate graph, since $\theta_{low} \leq \theta$, the approximate centrality of node u_{low} is never greater than θ (Lemma 2). Similarly, in the original graph, the estimate centrality of node u_{low} is never grater than θ (Lemma 3). The algorithm discards a node if (and only if) its approximate or estimated centrality is greater than θ. Therefore, the lowest centrality node u_{low} can never be pruned during the search process. $\square$

We now turn to the complexity of Sniper. Note that the previous approaches need $O(n^2)$ space and $O(n^3)$ time to compute the lowest centrality node.

Theorem 2 (Complexity of Sniper). *Sniper requires $O(n+m)$ space and $O(n^2+nm)$ time to compute the lowest centrality node.*

Proof. We first prove that Sniper requires $O(n+m)$ space. Sniper keeps the approximate graph and the original graph. In the approximate graph, since the number of nodes and edges are at most n and m, respectively, Sniper needs $O(n+m)$ space for the approximate graph; $O(n+m)$ space is required for keeping the original graph. Therefore, the space complexity of Sniper is $O(n+m)$.

Next, we prove that Sniper requires $O(n^2+nm)$ time. To identify the lowest centrality node, Sniper first updates the approximate graph and then computes approximate and exact centralities. Sniper needs $O(nm)$ time to update the approximate graph, since it requires $O(m)$ time to compute similarity for the added node against each node in the original graph. It requires $O(n^2+nm)$ time to compute the approximate and exact centralities since the number of nodes and edges are at most n and m, respectively. Therefore, Sniper requires $O(n^2+nm)$ time. $\square$

Theorem 2 shows, theoretically, that the space and time complexities of Sniper are lower in order than those of the previous approximate approaches. In practice, the search cost depends on the effectiveness of the approximation and estimation techniques used by Sniper. In the next section, we show their effectiveness by presenting the results of extensive experiments.

7 Experimental Evaluation

We performed experiments to demonstrate Sniper's effectiveness in a comparison to two annotation approaches: the **Zone** annotation scheme and the **Distant to zone** annotation scheme (abbreviated to **DTZ**). These were selected since they outperform the other embedding schemes on the contents of our dataset; the same result is reported in [13]. Zone and DTZ annotation have two parameters:

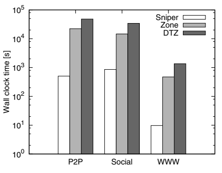


Fig. 1. Efficiency of Sniper

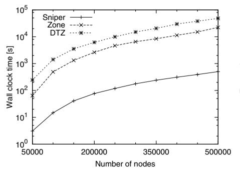
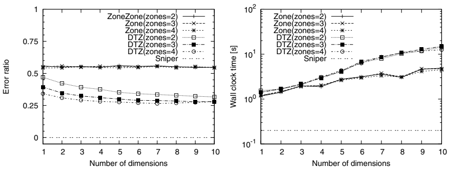


Fig. 2. Scalability of Sniper



(1) Error ratio (2) Wall clock time

Fig. 3. The results of the annotation approaches

zones and dimensions. Zones are divided regions of the entire graph, and dimensions are sets of zones[1]. Note that these approaches can compute centrality quickly at the expense of exactness.

We used the following three public datasets in the experiments: *P2P* [1], *Social* [2], and *WWW* [3]. They are a campus P2P network for file sharing, a free online social network, and web pages within the 'nd.edu' domain, respectively. We extracted the largest connected component from the real data, and we added single nodes one by one in the experiments.

We evaluated the search performance through wall clock time. All experiments were conducted on a Linux quad 3.33 GHz Intel Xeon server with 32GB of main memory. We implemented our algorithms using GCC.

7.1 Efficiency of Sniper

We assessed the search time needed for Sniper and the annotation approach. Figure 1 shows the efficiency of Sniper where the number of nodes are $500,000$ for P2P and Social, and $100,000$ for WWW. We also show the scalability of our approach in Figure 2; this figure shows the wall clock time as a function of the number of nodes. We show only the result of P2P in Figure 2 due to space limitations. These figures indicate Sniper's total processing time (both update and search time are included). We set the number of zones and the dimension parameter to 2 and 1, respectively. Note that, these parameter values allow the annotation approaches to estimate the lowest centrality node most efficiently.

These figures show that our method is much faster than the annotation approaches under all conditions examined. Specifically, Sniper is more than 110 times faster.

The annotation approaches require $O(n^2)$ time for computing centralities while Sniper requires $O(n' + m')$ time for computing approximate centralities. Even if Sniper computes the approximate centralities of all aggregate nodes to prune the nodes, this cost does not alter the search cost since approximate computations are effectively terminated. Sniper requires $O(n + m)$ time to compute

[1] To compute the centralities of all nodes by the annotation approaches, we sampled half pairs from all nodes, which is the same setting used in [13].

exact centralities for nodes that cannot be pruned through approximation. This cost, however, has no effect on the experimental results. This is because a significant number of nodes are pruned by approximation.

7.2 Exactness of the Search Results

One major advantage of Sniper is that it guarantees the exact answer, but this raises the following simple question: 'How successful are the previous approaches in providing the exact answer even though they sacrifice exactness?'.

To answer this question, we conducted comparative experiments on the annotation approaches. As the metric of accuracy, we used the error ratio, which is the error centrality value of the estimated lowest centrality node divided by the centrality value of the exact answer node. Figure 3-(1) shows the error ratio and the wall clock time of the annotation approaches with various parameter settings. The number of nodes is $10,000$ and the dateset used is P2P in these figures.

As we can see from Figure 3, the error ratio of Sniper is 0 because it identifies the lowest centrality node without fail. The annotation approaches, on the other hand, have much higher error ratios. Therefore, it is not practical to use the annotation approaches in identifying the lowest centrality node. Figure 3-(2) shows that Sniper greatly reduces the computation time even though it guarantees the exact answer. The efficiency of the annotation approaches depends on the parameters used.

Furthermore, the results show that the annotation approaches force a trade-off between speed and accuracy. That is, as the number of zones and dimensions parameters decreases, the wall clock time decreases but the error ratio increases. The annotation approaches are approximation techniques and so can miss the lowest centrality node. Sniper also computes approximate centralities, but unlike the annotation approaches, Sniper does not discard the lowest centrality node in the search process. As a result, Sniper is superior to the annotation approaches in not only accuracy, but also speed.

8 Conclusions

This paper addressed the problem of finding the lowest closeness centrality node from time-evolving graphs efficiently and exactly. Our proposal, Sniper, is based on two ideas: (1) It approximates the original graph by aggregating original nodes to compute approximate centralities efficiently, and (2) It terminates unnecessary distance computations early in finding the answer nodes, which greatly improves overall efficiency. Our experiments show that Sniper works as expected; it can find the lowest centrality node at high speed; specifically, it is significantly (more than 110 times) faster than existing approximate methods.

References

1. http://kdl.cs.umass.edu/data/canosleep/canosleep-info.html
2. http://snap.stanford.edu/data/soc-LiveJournal1.html
3. http://vlado.fmf.uni-lj.si/pub/networks/data/ND/NDnets.htm
4. Broder, A.Z., Glassman, S.C., Manasse, M.S., Zweig, G.: Syntactic clustering of the web. Computer Networks 29(8-13), 1157–1166 (1997)
5. Elmacioglu, E., Lee, D.: On six degrees of separation in dblp-db and more. SIGMOD Record 34(2), 33–40 (2005)
6. Garfield, E.: Citation Analysis as a Tool in Journal Evaluation. Science 178, 471–479 (1972)
7. Goldberg, A.V., Harrelson, C.: Computing the shortest path: search meets graph theory. In: SODA, pp. 156–165 (2005)
8. Leskovec, J., Kleinberg, J.M., Faloutsos, C.: Graph evolution: Densification and shrinking diameters. TKDD 1(1) (2007)
9. Nascimento, M.A., Sander, J., Pound, J.: Analysis of sigmod's co-authorship graph. SIGMOD Record 32(3), 8–10 (2003)
10. Newman: The structure and function of complex networks. SIREV: SIAM Review 45 (2003)
11. Ng, T.S.E., Zhang, H.: Predicting internet network distance with coordinates-based approaches. In: INFOCOM (2002)
12. Potamias, M., Bonchi, F., Castillo, C., Gionis, A.: Fast shortest path distance estimation in large networks. In: CIKM, pp. 867–876 (2009)
13. Rattigan, M.J., Maier, M., Jensen, D.: Using structure indices for efficient approximation of network properties. In: KDD, pp. 357–366 (2006)