Efficient Betweenness Maximization in Temporal Networks

Xijuan Liu
Hangzhou Dianzi University
Hangzhou, China
liuxijuan@hdu.edu.cn

Kejia Xu
University of New South Wales
Sydney, Australia
kejia.xu@unsw.edu.au

Lele Zhang
Zhejiang Gongshang University
Hangzhou, China
lelez.zjgsu@gmail.com

Haiyang Hu
Hangzhou Dianzi University
Hangzhou, China
huhaiyang@hdu.edu.cn

Ying Zhang
Common Prosperity Visualization and Policy Simulation Lab of Zhejiang Gongshang University
Hangzhou, China
ying.zhang@zjgsu.edu.cn

ABSTRACT

Betweenness centrality is a crucial metric widely used in network analysis tasks to measure centrality and identify the most central vertices. The betweenness centrality of a vertex is defined as the frequency with which the shortest paths between other vertices pass through it. In many applications, the primary objective is to find a set of vertices with the maximum betweenness centrality, referred to as the betweenness centrality maximization problem. Although this problem has been extensively studied in static networks, many real-world networks are not only large in scale but also incorporate temporal information. Therefore, solving this problem in temporal networks necessitates the consideration of additional criteria, and the computational cost is prohibitive compared to static networks. In this paper, we propose an efficient approximation algorithm, KTBCM, which provides high-quality estimates for the temporal betweenness centrality maximization problem. KTBCM first transforms the temporal network into an edge graph to mitigate the time-consuming process of comparing adjacent temporal information. Moreover, it reduces the size of the edge graph by employing the pruning technique. Then, KTBCM utilizes a single-path sampling strategy to accelerate the search for the shortest paths without compromising the quality of the result. Finally, experiments on 6 temporal networks demonstrate that KTBCM achieves results comparable to the exact algorithm while significantly reducing computational costs in large networks.

VLDB Workshop Reference Format:
Xijuan Liu, Kejia Xu, Lele Zhang, Haiyang Hu, and Ying Zhang. Efficient Betweenness Maximization in Temporal Networks. VLDB 2025 Workshop: LSGDA.

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

1 INTRODUCTION

In complex network analysis, centrality is a fundamental measure extensively used to identify central vertices [7]. Centrality measures can be applied to various tasks, such as epidemic research [8], network vulnerability analysis [19], and rout planning [23]. In this work, we consider the betweenness centrality, a commonly used centrality measure that quantifies the frequency of a vertex lies on the shortest paths between pairs of other vertices [15, 25]. Vertices with high betweenness centrality play a crucial role in detecting communities [14], controlling global information interaction flows [26], and clustering [17].

In many real-world applications, finding a group of vertices with high betweenness centrality is important [21, 28, 35]. In biological networks, we are more interested in the identity of whether a group of components can be the key to the functional activities of other components. In cyberattacks, defending a set of nodes that can impact the entire network leads to more efficient resistance against attacks [3]. Such tasks can be summarized as identifying a set of k vertices with the maximum betweenness centrality, referred to as the Betweenness Centrality Maximization problem (BCMP).

However, existing works only consider betweenness centrality maximization in static networks while ignoring the temporal information. Real-world systems are time-dependent, showing the interactions between vertices, such as stock trading markets and transportation systems. Compared to static networks, temporal networks provide a more comprehensive characterization of the network properties [34, 36, 37, 39]. To fill the gap, we propose a novel problem, the temporal Betweenness Centrality Maximization problem. However, incorporating temporal information makes the problem more challenging than in static networks. Additional criteria include the shortest temporal path (STP) criterion and the restless temporal path (RTP) criterion are involved in finding the shortest paths [20, 33]. The STP criterion requires timestamps on the path between two vertices in ascending order, with the minimum number of edges. The RTP criterion builds on the STP criterion by further requiring that the interval of consecutive timestamps fall within a predefined parameter $\lambda \in \mathbb{R}$. In the temporal network shown in Figure 1(a), following the STP and RTP criteria with $\lambda = 5$, we can obtain the shortest temporal paths between vertex v_1 and v_5 as shown in Figure 1(b) and 1(c), respectively, While the shortest

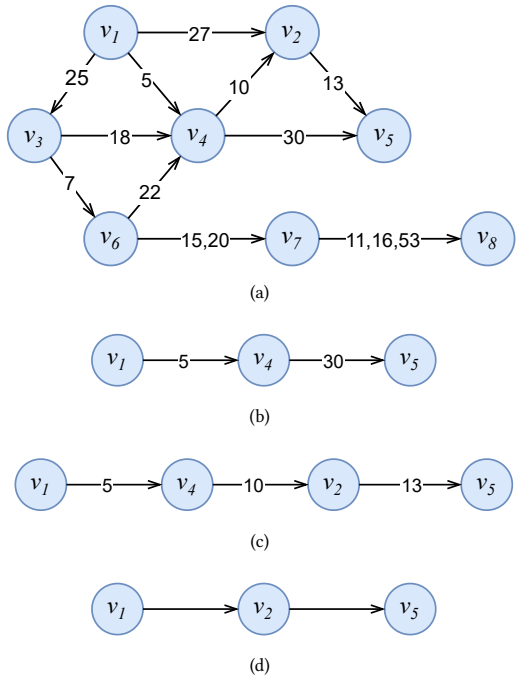


Figure 1: An example of searching the shortest paths in a directed temporal network under different criteria.

path between v_1 and v_5 in the corresponding static network is different as shown in Figure 1(d). Therefore, the methods for finding the shortest paths in static networks cannot be directly applied to temporal networks in an efficient way.

The BCMP is proved to be NP-hard and APX-complete [9, 13]. Research by Fink and Spoerhase demonstrate that using a greedy algorithm for maximizing the monotone-submodular objective function ensures an approximation ratio of $1 - 1/e$ [13]. However, the greedy algorithm requires at least $O(kmn)$ time, which is obviously impractical for large networks. Considering the relationship between finding the shortest paths and obtaining each vertex's betweenness centrality, the computational costs increase significantly when temporal information is added. For large temporal networks, such time complexity is prohibitive.

To address the above challenges, our intuition is to minimize the impact of the temporal information on finding the shortest paths. The STP and RTP criteria introduce additional constraints for identifying the shortest path, requiring multiple timestamp comparisons. Therefore, we transform the temporal network into an edge graph, where each edge is mapped to an edge vertex. During the transformation, edges between edge vertices are built iff the criteria are satisfied. By searching for the shortest paths in the edge graph, we eliminate redundant timestamp comparisons, significantly reducing computational costs. In greedy algorithms, calculating betweenness centrality requires obtaining all shortest paths passing through each vertex, which is time-consuming. Inspired by [28], we sample only one shortest path between two vertices to further accelerate the computation. Based on these optimizations, we propose the k Top Betweenness Centrality Maximization algorithm (KTBCM), the first efficient approximation algorithm for the temporal BCMP. Our main contributions are as follows:

- We introduce a novel problem, the temporal betweenness centrality maximization problem, and present an efficient approximation algorithm, KTBCM. To our best knowledge, this is the first paper to study the betweenness centrality maximization in temporal graph.
- To solve the problem efficiently, we transform the temporal network into an edge graph, significantly reducing the time costs on timestamp comparisons.
- We design a lossless pruning technique to reduce the scale of the edge graph and sample only one shortest path between vertex pairs within the edge graph to accelerate the search computation.
- Experiments conducted on 6 real-world temporal networks demonstrate that our algorithm achieves accuracy comparable to the exact algorithm, and provides orders of magnitude speedup compared to approximation algorithms.

The rest of this paper is organized as follows. In Section 2, we introduce the problem definition. In Section 3, we introduce our proposed algorithm Top-k-TBCM and present its theoretical analysis. Section 4 presents the experiments results over 6 datasets. We introduce the related works in Section 5 and conclude the paper in Section 6.

2 PRELIMINARIES

In this paper, we consider an unweighted direct temporal graph $T = (V, E)$, where V (resp. E) is the set of n vertices (resp. m direct edges). Each edge $e \in E$ can be represented as (u, v, t), where $u, v \in V$ and t is the corresponding timestamp. For each vertex v, we use $N(v) = \{(u, t) | v \in V, (v, u, t) \in E\}$ to denote the neighbours of v.

Definition 2.1 (Temporal Path). Given a temporal network T, $P_{s,z} = \langle e_1 = (s, v_1, t_1), e_2 = (u_2, v_2, t_2), ..., e_l = (v_l, z, t_l) \rangle$ is an l-edge temporal path from $s \in V$ to $z \in V$. The vertices in $P_{s,z}$ are ordered by increasing timestamps, *i.e.*, $t_1 \leq t_2 \leq ... \leq t_l$.

Note that, the method proposed in this paper can be easily extended to support the case with non-strict ascending timestamps, i.e., with $\leq$ constraint in a temporal path. Given a source vertex s and target vertex z, there are multiple temporal paths between the two vertices. To characterize the shortest path, we follow the previous settings in temporal betweenness centrality, and employ the Shortest Temporal Path (STP) and the Restless Temporal Path (RTP) criteria.

Definition 2.2 (Shortest Temporal Path). A shortest temporal path in T is a temporal path from s to z, denoted as $P^{sh}_{s,z}$, with the minimal length among all temporal paths from s to z.

Definition 2.3 (Restless Temporal Path). A restless temporal path in T is a shortest temporal path with the interaction time between any two consecutive edges does not exceed $\lambda \in \mathbb{R}$, denoted as $P^{rl}_{s,z}$.

For presentation simplicity, we use the STP by default. We use $\sigma^{sh}_{s,z}$ to denote the number of $P^{sh}_{s,z}$, and $\sigma^{sh}_{s,z}(v)$ to denote the number of $P^{sh}_{s,z}$ that pass through vertex v.

Definition 2.4 (Temporal Betweenness Centrality). Given a temporal graph $T(V, E)$ and a vertex $v \in V$, the temporal betweenness centrality of v is defined as

$$B(v) = \frac{1}{n(n-1)} \sum_{s,z \in V, s \neq z} \frac{\sigma_{s,z}^{sh}(v)}{\sigma_{s,z}^{sh}} \tag{1}$$

As discussed, in real-world applications, users may be interested in a group of vertices instead of individual ones. By extending the Define 1, we can compute the temporal betweenness centrality for a set of vertices $S \subseteq V$ as

$$B(S) = \frac{1}{n(n-1)} \sum_{s,z \in V, s \neq z} \frac{\sigma_{s,z}^{sh}(S)}{\sigma_{s,z}^{sh}}, \tag{2}$$

where $\sigma_{s,z}^{sh}(S)$ is the number of shortest temporal paths $P_{s,z}^{sh}$ that pass through at least one vertex in set S.

Problem statement. Given a temporal graph T and a positive integer k, the temporal betweenness centrality maximization problem (TBCMP) aims to find a set S^* of k vertices which has the largest temporal betweenness centrality, i.e.,

$$S^* = \underset{S \subseteq V : |S| \leq k}{\arg\max} \; B(S) \tag{3}$$

Theorem 2.5. *The objective function is a monotone and submodular function maximization problem.*

Proof. **Monotonicity**: Let $S_1 \subseteq S_2 \subseteq V$. According to the Equation 2, the betweenness centrality value depends on $\sigma_{s,z}^{sh}(S)$. Since $S_1 \subseteq S_2$, there are at least as many, and potentially more, vertices in S_2 that are passed through by shortest paths as in S_1. We can have: if $S_1 \subseteq S_2 \subseteq V$, then $B(S_1) \leq B(S_2)$.
Submodularity: We consider an vertex $v \in V \setminus S_2$, and show that $B(S_2 \cup \{v\}) - B(S_2) \leq B(S_1 \cup \{v\}) - B(S_1)$. As $\sigma_{s,z}^{sh}(S_1) \leq \sigma_{s,z}^{sh}(S_2)$, there will be some new paths pass through vertices in S_1 which are already pass through vertices in S_2 when adding vertex v. Then, the marginal gain of $B(S_2 \cup \{v\})$ will not be greater than that of $B(S_1 \cup \{v\})$. Therefore, $B(S_2 \cup \{v\}) - B(S_2) \leq B(S_1 \cup \{v\}) - B(S_1)$ holds.

□

3 SOLUTION

Due to the properties of the objective function, applying the greedy framework, i.e., iteratively selecting the vertex with the largest marginal gain, we can obtain a result with $1 - 1/e$ approximation ratio [13]. However, as discussed, computing temporal betweenness centrality exactly is time-consuming. There are some studies that aim to accelerate computing through approach approaches (e.g., [33]). While, it more focuses on computing the betweenness centrality for all vertices, and all the shortest paths are retrieved for each sample, which is not practical for the maximization task. As shown in the experiment, the performance is significantly affected.

3.1 Framework Overview

To solve TBCMP efficiently, we follow the sampling framework for betweenness centrality maximization in static graphs (e.g., [28]), which *i*) first generates a set of samples, *ii*) then iteratively select k vertices with the largest marginal gain over the samples. As the experimental results are shown in Table 1 for ONBRA, sampling the shortest paths is the most time-consuming part in temporal networks. To generate samples for TBCMP efficiently, we propose a graph transformation approach, which can reduce the cost of timestamp comparison for sampling. The overall framework is shown in Algorithm 1, denoted as KTBCM. It has three phases, i.e., graph transformation, sample generation, and seed computation. Note that, in this paper, we more focus on generating samples efficiently. For the sample size required, we follow the previous research, which is orthogonal to our study.

Algorithm 1: KTBCM

Input : A temporal network $T = (V, E)$, seed set size k, and the sample size M
Output: A seed set S of size k

```
G_e ← Graph-Transform(T);   /* graph transformation */;
PList ← Path-Find(T, G_e, M);   /* sample generation */;
S ← Seed-Selection(T, PList, k);   /* seed computation */;
return S
```

3.2 Graph Transformation

To avoid redundant comparison in the shortest path sampling, we transform the temporal network T into an edge graph G_e.

Vertex and edge transformation. Each vertex $v \in V$ is mapped to a V-vertex in G_e with a timestamp of 0, *i.e.*, $(v_i, v_i, 0)$. For any edge $e_i \in E$, we transform it into an E-vertex. For example, the edge between vertex v and u with a timestamp t is transformed into an E-vertex (v, u, t). Each V-vertex is only reachable by its neighboring E-vertices.

Edge generation. According to Definition 2.1, an edge is reachable when the timestamp is larger than that of the previous edge. For each E-vertex and V-vertex $(v, u, t) \in G_e$, we build an edge $((v, u, t_1), (u, w, t_2))$ from E(V)-vertex (v, u, t_1) to (u, w, t_2) iff $t_1 \leq t_2$.

When constructing the edge graph, there is a scenario where removing certain edges does not affect searching the shortest path. We refer to these edges as **meaningless**. If we can remove these edges, the search time could be significantly reduced.

Lemma 3.1. *If two temporal paths have the same vertices and order, the one with a larger timestamp is **meaningless** and can be removed when constructing the edge graph.*

Proof. Suppose there are E-vertices (v_i, v_j, t_1), (v_j, v_k, t_2) and $(v_j, v_k, t_3) \in G_e$, and $t_1 \leq t_2 \leq t_3$. Let $e_i = ((v_i, v_j, t_1), (v_j, v_k, t_2))$ be the edge from (v_i, v_j, t_1) to (v_j, v_k, t_2), $e_j = ((v_i, v_j, t_1), (v_j, v_k, t_3))$ be the edge from (v_i, v_j, t_1) to (v_j, v_k, t_3). We can have two temporal paths from vertex v_i and v_k through v_j in T: $P_{v_i,v_k} = \langle e_1 = (v_i, v_j, t_1), e_2 = (v_j, v_k, t_2)\rangle$ and $P'_{v_i,v_k} = \langle e'_1 = (v_i, v_j, t_1), e'_2 = (v_j, v_k, t_3)\rangle$. Suppose there is an E-vertex (v_k, v_q, t_4) connects (v_j, v_k, t_3) in G_e through an edge $e_k = ((v_j, v_k, t_3), (v_k, v_m, t_4))$. There are two paths from (v_i, v_j, t_1) and (v_k, v_q, t_4): $P_1 = \langle e_1 = ((v_i, v_j, t_1), (v_j, v_k, t_2), e_i), e_2 = ((v_j, v_k, t_2), (v_k, v_q, t_4), e_k)\rangle$ and $P_2 = \langle e'_1 = ((v_i, v_j, t_1), (v_j, v_k, t_3), e_j), e'_2 = ((v_j, v_k, t_3), (v_k, v_q, t_4), e_k)\rangle$. We can have e_j to be removed while maintaining the connection between E-vertex (v_i, v_j, t_1) and

Table 1: The runtime of ONBRA

Dataset	Initialize structures	Sample paths	Compute betweenness
CollegeMsg	14.127s	51.900s	1.020s
EmailEu	504.97s	2019.4s	4.491s
Mathoverflow	149.59s	1045.6s	0.923s

Algorithm 2: Graph-Transformation

Input : A temporal network $T = (V, E)$
Output: An edge graph $G_e = (V_e, E_e)$

1 $V_e, E_e \leftarrow \emptyset$;
2 **for each** $v \in V$ **do**
3 $V_e \leftarrow V_e \cup (v, v, 0)$;
4 **for each** $(v, u, t) \in E$ **do**
5 $V_e \leftarrow V_e \cup (v, u, t)$;
6 **for each** $(v, u, t) \in V_e$ **do**
7 Mark all vertices as not visited;
8 **for each** $(u, w, t') \in N(u)$ **do**
9 **if** $t \leq t'$ *and* w *is not visited* **then**
10 $E_e \leftarrow E_e \cup ((v, u, t), (u, w, t'))$;
11 Mark w as visited;
12 $G_e = (V_e, E_e)$;
13 **return** G_e

(v_k, v_q, t_4) with the same number of edges. And the vertex v_i can still reach v_q in the temporal network through v_j and v_k in order. □

Lines 6 to 11 of Algorithm 2 shows the process of generating edges in G_e: *i*) traverse each vertex in G_e and mark it as not-visited, *ii*) for each vertex in the neighbor vertex set $N(u)$, create an edge $\{(v, u, t), (u, w, t')\}$ iff $t \leq t'$. To adapt to the RTP criterion here, the constrain $t \leq t'$ in line 9 need to be modified to $t \leq t' \leq t + \lambda$.

3.3 Sample Generation

In this section, we introduce the Algorithm 3 that computes the temporal shortest paths in G_e and return the path set $PList$. Based on the analysis in the algorithm HEDGE, the algorithm samples $M = O(\frac{klog(n)}{\epsilon^2})$ pair of vertex [28]. In each iteration, we randomly select a pair of vertices (s, z) from T. The algorithm performs a BFS search for the shortest paths from the source E-vertex $(s, s, 0)$ to the target E-vertex $(z, z, 0)$. For each vertex $(v, u, t) \in G_e$, we initialize $dist_{v,u}(t)$ and a set $pre_{v,u}(t)$ to record the distance from the source E-vertex $(s, s, 0)$ and its predecessors, respectively. The $dist_{s,s}(0)$ is set to 0 for each source E-vertex. The BFS starts by pushing $(s, s, 0)$ into the queue. In each iteration, the algorithm pops a vertex (v, u, t) and checks if u is the target vertex. Then, the algorithm traverses neighbor vertices (v_n, u_n, t_n) of (v, u, t) with operations as follows: *i*) if u_n is the target vertex, the algorithm records the path by adding (v_n, u_n, t_n) into the predecessor set $pre_{z,z}(0)$; *ii*) if u_n is not the target vertex, the algorithm updates the distance with $dist_{v,u}(t) + 1$ and pushes it into the queue. The predecessor set is updated by checking the distance between (v, u, t) and (v_n, u_n, t_n). The algorithm keeps track of the shortest paths between s and z in the predecessor set $pre_{z,z}(0)$. If $pre_{z,z}(0)$ is not empty, the algorithm randomly samples one shortest path and adds it to the $PList$ set at the end of each iteration. In the sampling, vertices can be visited multiple times which is not allowed under the STP criterion.

Lemma 3.2. *The BFS in Algorithm 3 guarantees that no vertex will be repeated in the sampled temporal shortest path.*

Proof. Suppose there is a temporal shortest path from a source vertex s to a target vertex z that is not sampled by BFS: $P_{s,z} = \langle e_1 = (s, v_1, t_1), e_2 = (v1, v_2, t_2), e_3 = (v2, s, t_3), e_4 = (s, v_1, t_4), e_5 = (v1, v_2, t_5), e_6 = (v_2, z, t_5)\rangle$ with a cycle: $s \xrightarrow{t_1} v_1 \xrightarrow{t_2} v_2 \xrightarrow{t_3} s$. We observe that $P_{s,z}$ violates the Definition 2.2 as there are duplicate vertices. As BFS starts from the source vertex and expands the search scope layer by layer, the search will reach z directly after first reaching v_2. Furthermore, BFS utilizes a set to record vertices that have been visited, thereby preventing vertices duplication and the occurrence of cycles. The Algorithm 3 implements this mechanism by checking whether $dist_{v,u}(t)$ is a non-negative value. □

Next, we explain why it is sufficient to consider only one shortest path for the betweenness centrality calculation.

In the temporal network T, the probability of randomly selecting s and z is $\frac{1}{n(n-1)}$. According to Definition 2, the probability that at least one vertex in $P^{sh}_{s,z}$ belongs to the set S is $\frac{\sigma^{sh}_{s,z}(S)}{\sigma^{sh}_{s,z}}$. The probability of randomly selecting a shortest path containing vertices from any $S \subseteq V$ is:

$$Prob(P^{sh}_{s,z} \cap S \neq \emptyset) = \sum_{s,z \in V, s \neq z} \frac{1}{n(n-1)} B(S). \tag{4}$$

Suppose we search $n(n-1)$ shortest paths between s and z in the algorithm, the number of shortest paths passing through vertices in S will be equal to $B(S)$. Therefore, sampling only one shortest path is sufficient to calculate the temporal betweenness centrality of $S \subseteq V$.

3.4 Seed Computation

In this section, we describe the process of computing the seed set by traversing paths in $PList$. The aim is to find a set of k vertices that cover the largest number of shortest paths. A counter $C(v)$ and a path set are initialized to keep track of $\sigma^{sh}_{s,z}(v)$ and the set of paths covering v, respectively. For each vertex v in $PList$, we set $C(v) = 1$, indicating that the vertex covers one shortest path. The computing process is shown as follows and will iterate until the seed set has k vertices: At each step, we select the vertex v with the maximum

Algorithm 3: Path-Finding

Input : A temporal network $T = (V, E)$, an edge graph $G_e = (V_e, E_e)$ and the sample size M
Output: A set of shortest path $PList$

for $i \leftarrow 1$ ***to*** M **do**
  get a pair vertices (s, z) from V at random;
  // BFS from s
  for each $(v, u, t) \in V_e$ **do**
    $dist_{v,u}(t) \leftarrow -1$;
    $pre_{v,u}(t) \leftarrow \emptyset$;
  $Q \leftarrow$ empty queue;
  $dist_{s,s}(0) \leftarrow 0$;
  $Q.push(s, s, 0)$;
  while Q *is not empty* **do**
    $(v, u, t) \leftarrow Q.pop()$;
    if $u = z$ **then**
      break;
    for each $(v_n, u_n, t_n) \in N_{v,u}(t)$ **do**
      if $dist_{v_n,u_n}(t_n) \neq -1$ **then**
        $Q.push((v_n, u_n, t_n))$;
        $dist_{v_n,u_n}(t_n) \leftarrow dist_{v,u}(t) + 1$;
        if $u_n = z$ **then**
          $pre_{z,z}(0) \leftarrow pre_{z,z}(0) \cup (v_n, u_n, t_n)$;
      if $dist_{v_n,u_n}(t_n) = dist_{v,u}(t) + 1$ **then**
        $pre_{v_n,u_n}(t_n) \leftarrow pre_{v_n,u_n}(t_n) \cup \{v, u, t\}$;
  if $pre_{z,z}(0)$ *is not empty* **then**
    get a shortest path through pre at random;
    add the shortest to $PList$;
return $PList$

$C(v)$ and add it to the seed set. Once selected, the influence of v is removed. The counters $C(u)$ for vertices part of the same shortest path recorded in the path set as v are decremented by 1.

4 EXPERIMENTS

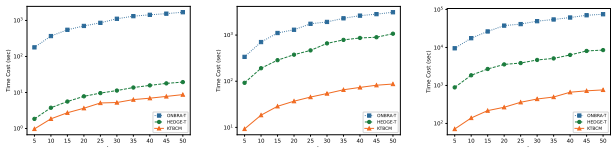
In this section, we conduct extensive experiments over 6 real-world graphs to demonstrate the performance of proposed techniques.

4.1 Experiment Setup

Datasets. The algorithm KTBCM is evaluated on 6 real-world networks. The statistic are summarized in Table 2. The largest network is Superuser, which contains 192 thousands vertices and 1.1 millions edges. All datasets are public available on SNAP[1]. We performed experiments in each network ten times and reported the average results.

Algorithms. Due to the lack of research on the TBCMP, we extend existing algorithms from static networks to temporal networks. We compare the proposed approximation algorithm KTBCM with the following algorithms: *i*) a greedy algorithm using exact computations of betweenness centrality, denoted as EXHAUST [13]; *ii*) an approximate algorithm ONBRA for computing temporal betweenness centrality, which consider the STP and RTP criterion [33]; *iii*) state-of-the-art approximation algorithm HEDGE computing BCMP in large static network [28]. The extended algorithms are denoted as EXHAUST-T, ONBRA-T and HEDGE-T. We evaluate the effectiveness of our algorithm by comparing it to the exact algorithm EXHAUST-T and assess the efficiency against the approximate algorithm ONBRA-T and HEDGE-T. All algorithms were implemented in C++ and complied with GCC 7.5.0 using the -03 optimization flag. Experiments were run on a Linux server equipped with an Intel(R) Xeon(R) Gold 5218R CPU (2.10GHz) processor and 64 GB of RAM.

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



(a) HighSchool2012 (b) CollegeMsg (c) EmailEu

Figure 2: Runtime on small networks.

4.2 Effectiveness Evaluation

To evaluate the effectiveness, we compared our algorithm with the algorithm EXHAUST-T. Due to the limited scalability of EXHAUST-T, experiments were conducted on three small networks, HighSchool2012, CollegeMsg and EmailEu. The seed size k is varied from 10 to 50 in steps of 10 to access the scalability. The values of betweenness centrality are normalized by $\frac{1}{n(n-1)}$, where n is the number of vertices in each network. As shown in Table 3, KTBCM achieves achieves accuracy comparable to EXHAUST-T. Although KTBCM is an approximate algorithm, it can still maintain high accuracy and scalability as the value of k increases. Since exact computation of betweenness centrality algorithms are usually much slower than approximate algorithms, we do not compare the running times of EXHAUST-T and KTBCM.

4.3 Efficiency Evaluation

We compare KTBCM with approximation algorithms HEDGE -T and ONBRA-T to evaluate the efficiency. KTBCM uses the same sampling strategy as HEDGE-T: sampling only one shortest path. In contrast, ONBRA-T obtains all shortest paths and then randomly selects one. For fair comparisons, all algorithms use the same sampling size. As shown in Figure 2, KTBCM is approximately 10 times faster than HEDGE-T, demonstrating the efficiency of transforming the temporal network into an edge graph for runtime reduction. Compared to ONBRA-T, KTBCM is 2 orders of magnitude faster on HighSchool2012 and EmailEu, indicating that sampling only one shortest path can achieve significant acceleration. According to the results, we found that as k increases, our algorithm's running time increases linearly with the value of k. Therefore, we compared the runtime on large networks when $k = 10$. As shown in Table

Table 2: Statistics of datasets

Dataset	Nodes	Edges	Granularity	Timespan
HighSchool2012	180	45K	20 sec	7 (days)
CollegeMsg	1.9K	59.8K	1 sec	193 (days)
EmailEu	986	332K	1 sec	803 (days)
Mathoverflow	24.8K	390K	1 sec	6.4 (years)
Askubuntu	157K	727K	1 sec	7.2 (years)
Superuser	192K	1.1M	1 sec	7.6 (years)

Table 3: KTBCM vs. EXHAUST-T: centralities

Dataset	k	EXHAUST-T	KTBCM
HighSchool2012	10	0.193	0.186
	20	0.274	0.269
	30	0.318	0.314
	40	0.347	0.342
	50	0.367	0.357
CollegeMsg	10	0.193	0.186
	20	0.274	0.269
	30	0.318	0.314
	40	0.347	0.342
	50	0.367	0.357
EmailEu	10	0.286	0.256
	20	0.396	0.355
	30	0.464	0.419
	40	0.512	0.470
	50	0.548	0.516

4, HEDGE-T and ONBRA-T struggle to complete the computation, whereas KTBCM can still complete the task within a reasonable time frame.

5 RELATED WORK

5.1 Betweenness Centrality Computation

In static networks. The naive exact algorithm to compute the betweeness centrality is time-consuming, requiring $O(n^3)$ time [15]. The best-known exact algorithm, proposed by Brandes, employed an accumulation technique and a traversal algorithm to compute single-source shortest paths, significantly reducing the time complexity to $O(mn)$ [5]. Erdös et al. utilize Brandes's algorithm and partition large networks into sub-networks to parallelize the computation of betweenness centrality within these sub-networks [10]. This algorithm archives a k-fold speedup [10]. However, the computational cost of these exact algorithms remains prohibitive for large real-world networks. Recently, research have introduced various sampling-based approximation algorithms [2, 4, 16, 18, 22, 32]. Bader et al. propose an adaptive sampling algorithm that also builds upon Brandes's algorithm, estimating betweenness centrality by computing single-source shortest paths for a sampled vertex subset [1]. Geisberger et al. introduce a scaling function, which provides higher approximation accuracy for less important nodes, making it suitable for large networks [16]. The time complexity of sampling algorithms generally depends on the sample size, usually achieving time complexity with $O(km)$ [32]. As previously discussed, both exact and approximation algorithms involve computing shortest paths, they cannot be directly applied to temporal networks.

In temporal networks. Kim and Anderson are the first to extend the betweenness centrality criteria to dynamic networks by transforming the dynamic network into time-ordered networks [24]. However, algorithms designed for dynamic networks are not applicable to temporal networks as the shortest paths are computed within a fixed timestamp in dynamic networks [33]. Buß et al. extended Brandes's algorithm to temporal networks based on various criteria, including the foremost and fastest paths, along with the STP criterion discussed in this paper [6]. Yet, this algorithm is highly inefficient, with a time complexity of $O(n^3t^2)$, where t is the number of time steps [6]. Zhang et al.'s work achieves high-quality results with multiple criteria considered through a novel recursive temporal dependency formulation and a network compression technique [41]. However, this algorithm does not process the RTP criterion. Conversely, ONBRA, proposed by Santoro and Sarpe, handles both the STP and RTP criteria [33]. ONBRA is the most related algorithm to our work and is the first to compute betweenness centrality in temporal networks based on sampling [33]. Compared to exact algorithms, ONBRA significantly accelerate the computation [33].

5.2 BCMP Computation In Static Networks

The earliest studies on centrality maximization prove that the centrality measure of an individual vertex can be extended to a set of vertices [11]. It provides new insight into the importance of groups within a network [11]. Puzis et al. introduce the BCMP, which involves identifying the k vertices in the network with the maximum betweenness centrality [30]. They prove that BCMP is NP-hard and demonstrate that using a greedy algorithm can achieve an approximation ratio of $1 - 1/e$, which is superior to that achieved by heuristic algorithms [30, 31]. The greedy algorithm works as follows: first starts with an empty set and iteratively adds the vertex that maximizes the betweenness centrality of the set at each step until the set contains k vertices [12, 13]. Such greedy algorithms require computing all shortest paths passing through each vertex, making the running time dependent on the size of the network. When the network is massive, greedy algorithms become impractical to complete within a reasonable timeframe.

Table 4: Runtime on large networks ($k = 10$)

Dataset	HEDGE-T	ONBRA-T	KTBCM
Mathoverflow	18101.8s	14182.4s	366.4s
Askubuntu	220980s	27725.3s	689.6s
Superuser	304167s	56038.9s	1417.1s

Consequently, previous research focus on developing approximation algorithms to deliver estimations within acceptable runtime [9]. Yoshida first explores BCMP in static networks, introducing a sampling-based algorithm called TOP-k-ABC for adaptive betweenness centrality measure [38]. TOP-k-ABC has a time complexity of $o((n + m + hlogn)logn/\epsilon^2)$, where h depends on the size of the set of shortest paths between any pair of vertices [38]. It computes all possible shortest paths between any pair of vertices and iteratively selects the top k vertices with the highest betweenness centrality based on the weights on the shortest paths [38]. The high-quality output is guaranteed by sampling $O(logn/\epsilon^2)$ sets of vertices [38]. HEDGE, another sampling-based algorithm, is currently considered state-of-the-art [28]. It samples only one shortest path between a pair of vertices to reduce sampling time, ensuring results with a $1 - 1/e - \epsilon$ approximation ratio [28]. Mumtaz et al. introduced a progressive random sampling approach PS, which sequentially selects nodes with higher frequency in the hyper-edge set to add to the result set [29]. However, when applied to large networks, PS does not outperform HEDGE in terms of accuracy. Additionally, Zhang et al. propose a sketch based method to efficiently find a set of results [40]. Veremyev et al. utilize a mixed integer programming solver for the BCMP, but the time complexity is similar to that of exact algorithms, resulting in low efficiency [35]. Li et al. investigate the BCMP when giving a set of query keywords [27]. It is evident that there are no known approximate algorithms designed for the temporal betweenness centrality maximization problem.

6 CONCLUSION

In this paper, we propose a novel problem: the Temporal Betweenness Centrality Maximization Problem, and present an algorithm named KTBCM. KTBCM incorporates a graph transformation approach with pruning technique and a one shortest path sampling strategy. Extensive experiments with 6 real-world networks demonstrate the effectiveness and efficiency of our proposed algorithm. In the future, we plan to investigate how to extend the algorithm to other types of networks.

REFERENCES

[1] David A Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. 2007. Approximating betweenness centrality. In *Algorithms and Models for the Web-Graph: 5th International Workshop, WAW 2007, San Diego, CA, USA, December 11-12, 2007. Proceedings 5*. Springer, 124–137.

[2] Miriam Baglioni, Filippo Geraci, Marco Pellegrini, and Ernesto Lastres. 2014. Fast exact and approximate computation of betweenness centrality in social networks. In *State of the Art Applications of Social Network Analysis*. Springer, 53–73.

[3] Ettore Bompard, Di Wu, and Fei Xue. 2010. The concept of betweenness in the analysis of power grid vulnerability. In *2010 Complexity in Engineering*. IEEE, 52–54.

[4] Michele Borassi and Emanuele Natale. 2019. KADABRA is an adaptive algorithm for betweenness via random approximation. *Journal of Experimental Algorithmics (JEA)* 24 (2019), 1–35.

[5] Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. *Journal of mathematical sociology* (2001).

[6] Sebastian Buß, Hendrik Molter, Rolf Niedermeier, and Maciej Rymar. 2020. Algorithmic Aspects of Temporal Betweenness. *arXiv: Data Structures and Algorithms,arXiv: Data Structures and Algorithms* (Jun 2020).

[7] Kousik Das, Sovan Samanta, and Madhumangal Pal. 2018. Study on centrality measures in social networks: a survey. *Social network analysis and mining* 8 (2018), 1–11.

[8] Anthony H Dekker. 2013. Network centrality and super-spreaders in infectious disease epidemiology. In *20th International Congress on Modelling and Simulation (MODSIM2013)*. Citeseer, 331–337.

[9] Shlomi Dolev, Yuval Elovici, Rami Puzis, and Polina Zilberman. 2009. Incremental deployment of network monitors based on group betweenness centrality. *Inform. Process. Lett.* 109, 20 (2009), 1172–1176.

[10] Dóra Erdős, Vatche Ishakian, Azer Bestavros, and Evimaria Terzi. 2015. A divide-and-conquer algorithm for betweenness centrality. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 433–441.

[11] Martin G Everett and Stephen P Borgatti. 1999. The centrality of groups and classes. *The Journal of mathematical sociology* 23, 3 (1999), 181–201.

[12] Uriel Feige. 1998. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)* 45, 4 (1998), 634–652.

[13] Martin Fink and Joachim Spoerhase. 2011. Maximum betweenness centrality: approximability and tractable cases. In *WALCOM*. Springer, 9–20.

[14] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* 486, 3-5 (2010), 75–174.

[15] Linton C Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* (1977), 35–41.

[16] Robert Geisberger, Peter Sanders, and Dominik Schultes. 2008. Better approximation of betweenness centrality. In *2008 Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 90–100.

[17] Michelle Girvan and Mark EJ Newman. 2002. Community structure in social and biological networks. *Proceedings of the national academy of sciences* 99, 12 (2002), 7821–7826.

[18] Mostafa Haghir Chehreghani. 2013. An efficient algorithm for approximate betweenness centrality computation. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 1489–1492.

[19] Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. 2002. Attack vulnerability of complex networks. *Physical review E* 65, 5 (2002), 056109.

[20] Petter Holme and Jari Saramäki. 2012. Temporal networks. *Physics Reports* 519, 3 (2012), 97–125.

[21] Swami Iyer, Timothy Killingback, Bala Sundaram, and Zhen Wang. 2013. Attack robustness and centrality of complex networks. *PloS one* 8, 4 (2013), e59613.

[22] Riko Jacob, Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters, and Dagmar Tenfelde-Podehl. 2005. Algorithms for centrality indices. *Network analysis: Methodological foundations* (2005), 62–82.

[23] Aisan Kazerani and Stephan Winter. 2009. Can betweenness centrality explain traffic flow. In *12th AGILE international conference on geographic information science*. Germany: Leibniz Universität Hannover, 1–9.

[24] Hyoungshick Kim and Ross Anderson. 2012. Temporal node centrality in complex networks. *Physical Review E* 85, 2 (2012), 026107.

[25] Ilkka Kivimäki, Bertrand Lebichot, Jari Saramäki, and Marco Saerens. 2016. Two betweenness centrality measures based on randomized shortest paths. *Scientific reports* 6, 1 (2016), 19668.

[26] Nicolas Kourtellis, Tharaka Alahakoon, Ramanuja Simha, Adriana Iamnitchi, and Rahul Tripathi. 2013. Identifying high betweenness centrality nodes in large social networks. *Social Network Analysis and Mining* 3 (2013), 899–914.

[27] Xiao Li, Yanping Wu, Xiaoyang Wang, Zhengyi Yang, Wenjie Zhang, and Ying Zhang. 2024. Keyword-Based Betweenness Centrality Maximization in Attributed Graphs. In *Australasian Database Conference*. Springer, 209–223.

[28] Ahmad Mahmoody, Charalampos E Tsourakakis, and Eli Upfal. 2016. Scalable betweenness centrality maximization via sampling. In *Proceedings of the 22nd*

ACM SIGKDD international conference on knowledge discovery and data mining. 1765–1773.

[29] Sara Mumtaz and Xiaoyang Wang. 2017. Identifying top-k influential nodes in networks. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2219–2222.

[30] Rami Puzis, Yuval Elovici, and Shlomi Dolev. 2007. Fast algorithm for successive computation of group betweenness centrality. *Physical Review E* 76, 5 (2007), 056709.

[31] Rami Puzis, Yuval Elovici, and Shlomi Dolev. 2007. Finding the most prominent group in complex networks. *AI communications* 20, 4 (2007), 287–296.

[32] Matteo Riondato and Evgenios M Kornaropoulos. 2014. Fast approximation of betweenness centrality through sampling. In *Proceedings of the 7th ACM international conference on Web search and data mining*. 413–422.

[33] Diego Santoro and Ilie Sarpe. 2022. Onbra: Rigorous estimation of the temporal betweenness centrality in temporal networks. In *Proceedings of the ACM Web Conference 2022*. 1579–1588.

[34] Zhiyang Tang, Yanping Wu, Xiangjun Zai, Chen Chen, Xiaoyang Wang, and Ying Zhang. 2025. Efficient Temporal Simple Path Graph Generation. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 3589–3601.

[35] Alexander Veremyev, Oleg A Prokopyev, and Eduardo L Pasiliao. 2017. Finding groups with maximum betweenness centrality. *Optimization Methods and Software* 32, 2 (2017), 369–399.

[36] Yanping Wu, Renjie Sun, Xiaoyang Wang, Dong Wen, Ying Zhang, Lu Qin, and Xuemin Lin. 2024. Efficient maximal frequent group enumeration in temporal bipartite graphs. *PVLDB* (2024).

[37] Yanping Wu, Renjie Sun, Xiaoyang Wang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2024. Efficient maximal temporal plex enumeration. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 3098–3110.

[38] Yuichi Yoshida. 2014. Almost linear-time algorithms for adaptive betweenness centrality using hypergraph sketches. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1416–1425.

[39] Jianke Yu, Hanchen Wang, Xiaoyang Wang, Zhao Li, Lu Qin, Wenjie Zhang, Jian Liao, Ying Zhang, and Bailin Yang. 2024. Temporal Insights for Group-Based Fraud Detection on e-Commerce Platforms. *IEEE Transactions on Knowledge and Data Engineering* (2024).

[40] Lele Zhang, Yanping Wu, Jinghao Wang, Shiyuan Liu, Chen Chen, and Xiaoyang Wang. 2023. Efficient algorithm for maximizing betweenness centrality in large networks. In *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. IEEE, 647–654.

[41] Tianming Zhang, Yunjun Gao, Jie Zhao, Lu Chen, Lu Jin, Zhengyi Yang, Bin Cao, and Jing Fan. 2024. Efficient Exact and Approximate Betweenness Centrality Computation for Temporal Graphs. In *Proceedings of the ACM on Web Conference 2024*. 2395–2406.