Front. Comput. Sci., 2025, 19(2): 192601
https://doi.org/10.1007/s11704-023-3424-y

RESEARCH ARTICLE

Labeling-based centrality approaches for identifying critical edges on temporal graphs

Tianming ZHANG (✉)[1], Jie ZHAO[1], Cibo YU[1], Lu CHEN[2], Yunjun GAO[2], Bin CAO (✉)[1], Jing FAN[1], Ge YU[3]

1 School of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China
2 School of Computer Science and Technology, Zhejiang University, Hangzhou 310013, China
3 Department of Computer Science, Northeastern University, Shenyang 110819, China

Abstract Edge closeness and betweenness centralities are widely used path-based metrics for characterizing the importance of edges in networks. In general graphs, edge closeness centrality indicates the importance of edges by the shortest distances from the edge to all the other vertices. Edge betweenness centrality ranks which edges are significant based on the fraction of all-pairs shortest paths that pass through the edge. Nowadays, extensive research efforts go into centrality computation over general graphs that omit time dimension. However, numerous real-world networks are modeled as temporal graphs, where the nodes are related to each other at different time instances. The temporal property is important and should not be neglected because it guides the flow of information in the network. This state of affairs motivates the paper's study of edge centrality computation methods on temporal graphs. We introduce the concepts of the label, and label dominance relation, and then propose multi-thread parallel labeling-based methods on OpenMP to efficiently compute edge closeness and betweenness centralities w.r.t. three types of optimal temporal paths. For edge closeness centrality computation, a time segmentation strategy and two observations are presented to aggregate some related temporal edges for uniform processing. For edge betweenness centrality computation, to improve efficiency, temporal edge dependency formulas, a labeling-based forward-backward scanning strategy, and a compression-based optimization method are further proposed to iteratively accumulate centrality values. Extensive experiments using 13 real temporal graphs are conducted to provide detailed insights into the efficiency and effectiveness of the proposed methods. Compared with state-of-the-art methods, labeling-based methods are capable of up to two orders of magnitude speedup.

Keywords temporal graph, closeness centrality, betweenness centrality, temporal path

Received May 23, 2023; accepted October 19, 2023

E-mail: tmzhang@zjut.edu.cn; bincao@zjut.edu.cn

1 Introduction

Critical edge identification is a cornerstone for analyzing the structure, function, and behavior of complex networks. It has attracted considerable attention in the last decades. There are many centrality metrics (such as closeness centrality, betweenness centrality, eigenvector centrality, and Katz centrality) for characterizing the importance of edges in the graph, among which edge closeness centrality (ECC) and edge betweenness centrality (EBC) are the most widely studied. ECC and EBC are based on the concept of shortest paths, and are relevant with respect to the flux of information. The closeness centrality of an edge, which is extended by the vertex closeness centrality proposed in [1], is the reciprocal of the sum of the shortest distances from the edge to all the other vertices. Typically it judges how fast information will spread from an edge in a network to all other vertices. The betweenness centrality of an edge is defined as the fraction of the shortest paths that pass through the edge between pairs of nodes in the graph. It was first proposed by Newman et al. [2] as a way to extract community structures. An edge with a high edge betweenness centrality value is usually regarded as a bridge-like connector, and the removal of which may affect the connection between communities.

The two centrality metrics have many application domains, such as sensor and communication networks, social networks, engineering mechanics, and water distribution networks, etc. In sensor and communication networks, identifying and removing critical edges with high ECC or EBC values can truncate important communication links, and further control message transmission. In social networks, ECC and EBC can be employed to target influential individuals or groups for marketing or outreach purposes. In engineering mechanics, EBC is used as a failure predictor for two-dimensional fuse network models of load transmission in structurally disordered materials [3]. In water distribution networks, ECC and EBC are utilized as a tool for hydraulic modeling for water distribution network analysis and management [4].

Extensive research efforts go into centrality computation over general graphs that omit time dimension. However,

numerous real-world networks change over time and are modeled as temporal graphs, where the nodes are related to each other at different time instances. Representing the network as a temporal graph allows us to capture the time-related nature of the interactions. In a social network such as Twitter or Facebook, individuals interact with each other at specific time points. In a gene regulatory network, regulatory interactions between genes vary over time and are influenced by different internal and external factors, such as environmental conditions or disease states. The temporal property is important and should not be neglected because it guides the flow of information in the network. For example, a diffusion process on the network, such as the spread of rumors or diseases, should follow the temporal path that respects the timing order of the interactions. Hence, in this paper, we address how to efficiently compute temporal edge closeness and betweenness centrality of all edges based on temporal paths.

Example 1 Figure 1 illustrates a wireless sensor network that monitors physical or environmental conditions, such as temperature, sound, and pressure. Sensors are nodes and the actions of scheduled sending data to other sensors or receiving data from other sensors are denoted as edges. The timestamp attached with an edge represents send or receive time of data transmission. Calculating ECC or EBC values allows us to achieve high-quality service in wireless sensor networks by evaluating relationships between entities of the network (i.e., edges), thus controlling information flow, message delivery, and energy dissipation among nodes [5].

Some existing works [6,7] handle a temporal graph as a series of static snapshots, and then separately compute centrality values on each snapshot. While in this way, temporal paths between snapshots are not taken into account, leading to the overestimation of the diffusion of information in the network [8]. The temporal character of networks poses new challenges to efficiently computing ECC and EBC. (i) Different notions of optimal paths [9], such as shortest temporal path, earliest temporal path, and fastest temporal path, are introduced while working with temporal graphs. The concept of the classic shortest path is defective for ECC and EBC computation in a temporal graph, as the temporal information determines the order of events along a path. (ii) Temporal ECC and EBC computation is costly because it requires considering an excessive number of temporal paths from any edge/node at any time instance to any node at any time instance. Traditional greedy-based method for finding shortest paths in general graphs is inapplicable to temporal graphs as subpaths of a "shortest" path may not be "shortest" when computing optimal temporal paths.

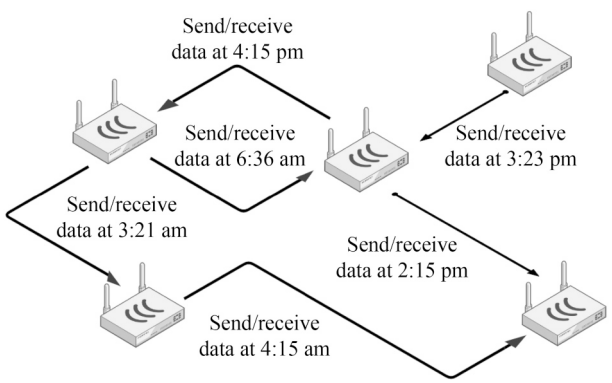


Fig. 1 An example of a wireless sensor network

To solve the above challenges, we propose labeling-based methods to compute ECC and EBC values w.r.t. three types of optimal temporal paths (i.e., shortest temporal path, earliest temporal path, fastest temporal path). The label records the necessary time information and is used for temporal path extension, and we define the label dominance relation to filter out superfluous and alternative labels, and thus reduce the number of potential paths for the extension. For ECC computation, we found that the sets of optimal temporal paths from some temporal edges are equal, hence we present a time segmentation strategy and two observations to aggregate these edges and calculate them together. For EBC computation, we propose temporal edge dependency formulas and a labeling-based forward-backward scanning strategy. In the forward scanning, we generate the labels and then progressively compute optimal temporal paths. In the backward scanning, starting from the optimal temporal paths with the maximum length, we iteratively accumulate the contribution of the subpaths to their containing edges. Furthermore, in order to reduce memory consumption and improve efficiency, we propose a temporal edge compression rule to compress and further jointly calculate the edges that have closely related EBC values. In addition, considering that the ECC and EBC calculations for each edge are relatively independent, we devise multithreading implementations of our approaches on OpenMP for higher parallelism. In brief, our contributions are summarized as follows.

- We introduce the concepts of the label, and label dominance relation, and then propose labeling-based methods to efficiently compute ECC and EBC values of all edges w.r.t. three types of optimal temporal paths (i.e., shortest temporal path, earliest temporal path, fastest temporal path).
- For ECC computation, a time segmentation strategy, and two observations are presented to aggregate edges from which the sets of optimal temporal paths are equal. For EBC computation, temporal edge dependency formulas and a labeling-based forward-backward scanning strategy are proposed to iteratively accumulate EBC values.
- To reduce memory occupation and improve efficiency, a compression rule is proposed to compress and further jointly calculate the edges whose EBC values are closely related. In addition, multithread implementations on OpenMP are devised for higher parallelism.
- Extensive experiments on 13 real-world temporal networks show the efficiency and effectiveness of the proposed labeling-based methods. Particularly, compared with the state-of-the-art, labeling-based methods are capable of up to two orders of magnitude speedup. The case study provides insights that temporal

ECC and EBC are useful for identifying the critical edges in information diffusion.

The rest of the paper is organized below. Section 2 discusses the related literature. Section 3 introduces the concepts and formulates the problem. Section 4 and Section 5 propose the techniques for ECC and EBC computation, respectively. The experimental results are reported in Section 6. Section 7 concludes the paper and discusses future work.

2 Related work

2.1 Closeness centrality computation

A plethora of papers have studied vertex closeness centrality in general networks, i.e. networks that ignore time dimension. Saxena [10] offered a comprehensive survey. A direct approach is to perform Breadth-First Search (BFS) on unweighted graphs and use Dijkstra algorithm on weighted graphs, which cost $O(nm)$ and $O(nm+n^2\log n)$, where n and m are the number of vertices and edges, respectively. To improve performance, researchers propose approximate and top-k closeness centrality computation algorithms. For example, Wang et al. [11] utilized Hoffding inequality [12] to ensure that the estimated values have a small deviation. Okamoto et al. [13] learned the idea of PageRank and proposed TopRank, which sorted the importance of top-k vertices, not all vertices. While these methods are only for undirected graphs. A near-linear-time algorithm for estimating closeness centrality values was presented in [14]. Whereas, it did not provide worst-case theoretical approximation guarantees for directed graphs. Olsen et al. [15] employed intermediate results to minimize the closeness centrality computation time of top-k vertices, but the method needs $O(n+m)$ additional space.

Recently, researchers have acknowledged that the dynamic character of networks impacts the centrality values, and thus some pieces consider closeness centrality on dynamic networks. In summary, most studies [6,16,17] represented dynamic networks as a sequence of snapshots (i.e., general networks), and then analyzed each snapshot separately or incrementally. The time dimension and temporal paths are essentially not taken into account. Crescenzi et al. [18] proposed a sampling-based approximation algorithm for the temporal closeness centrality. Oettershagen et al. [19] proposed TC-ALL (we called TGC in the experimental comparison) for computing the exact temporal closeness values of all vertices. It adopted the pruning framework introduced by Bergamini et al. [20] and computed the minimum duration paths in temporal graphs instead of BFS in static graphs.

2.2 Betweenness centrality computation

In the literature, numerous research is conducted to investigate vertex betweenness centrality over general graphs. As early as 2001, Brandes' algorithm [21] was designed. Then Erdős et al. [22] propose Brandes++, which adopted a divide-and-conquer strategy and utilized graph structure to accelerate calculation. Sariyüce et al. [23] compressed some special vertices, divided the graph into multiple subgraphs using joint points and bridge edges, and then calculated the betweenness centrality values of vertices of subgraphs. Baglion et al. [24] employed the graph topological features to accelerate computation. In addition, some variants of EBC are investigated. For instance, Kanwar et al. [25] propose a variant of EBC called BC_{DCN}. De Meo et al. [26] defined κ-path and calculated EBC values based on κ-path. To further improve efficiency, researchers proposed approximate calculation methods based on sampling (e.g., vertex sampling [27] and shortest path sampling [28,29]). However, these techniques applied to general graphs cannot be directly extended to temporal graphs, because temporal information is ignored and path computation in the temporal graph does not meet the optimal substructure property, which is the core theory for centrality computation on general graphs.

Some papers [30–35] are concerned with the study of vertex betweenness centrality computation on dynamic graphs or graph streams. The common idea of these works is to exploit auxiliary structures or reuse intermediate results to support incremental updates. Nevertheless, these works usually regard the temporal graph as a set of graph snapshots and do not consider the time dimension. For temporal graphs, three optimal temporal paths (i.e., earliest arrival path, latest departure path, fastest path) were explored in [9]. Tsalouchidou et al. [7] proposed a temporal path type that combined path length and temporal distance, and calculated betweenness centrality values on the static window with a fix-length. In essence, it is based on the optimal substructure properties of general graphs or the static extension. Buß et al. [36] introduced TGB, which constructed the predecessor graph to satisfy the optimal substructure property, and then extended Brandes' theory to the predecessor graph for calculation, but the proposed algorithm cannot be applied to the earliest and fastest temporal paths, and the predecessor graph structure consumes a large amount of memory, as verified in our experiments.

3 Preliminaries

In many real-life networks, such as email networks, epidemic networks, and online social networks, messages are sent at a point in time, or contacts between individuals are transient. Hence we define the temporal graph below, following literature [7,8,36].

Definition 1 (Temporal graph). A directed temporal graph is defined as $G=(V,E,T)$, where V, E, and T are the sets of the vertices, the temporal edges, and the timestamps, respectively. Specifically, a temporal edge $e=(u,v,t)$ represents an instantaneous event from u to v taking place at time $t\in T$.

This modeling allows capturing the dynamics and interactions of graphs over time, making it easier to analyze and respond to various scenarios and events. With the sensor network as an example, the temporal aspect comes into play when capturing how the sensors' connections change over time. The importance of edges, which reflects how much influence a particular edge has on the overall information flow within the sensor network, can be evaluated by the number of network transitions.

In the following, we focus on un-weighted temporal graphs,

but the proposed techniques can be easily extended to temporal graphs with non-negative weights on the temporal edges.

Let $E_{out}(v)$ and $E_{in}(v)$ refer to the sets of v's outgoing temporal edges and incoming temporal edges, respectively. Let $T_{out}(v) = \{t_w | (v,w,t_w) \in E_{out}(v)\}$ and $T_{in}(v) = \{t_v | (u,v,t_v) \in E_{in}(v)\}$ be the sets of distinct timestamps of v's outgoing temporal edges and v's incoming temporal edges, respectively. $E(u,v)$ is the edge set from u to v; $T(u,v) = \{t_v | (u,v,t_v) \in E(u,v)\}$ is the set of distinct timestamps from u to v. Table 1 summarizes the symbols frequently used in this paper.

Definition 2 (E-Temporal path). An e-temporal path from $e_1 = (v_0, v_1, t_1)$ to v_n is a sequence of consecutive edges $e_1 = (v_0, v_1, t_1)$, $e_2 = (v_1, v_2, t_2)$, …, $e_n = (v_{n-1}, v_n, t_n)$, denoted by $p(e_1, v_n) = v_0 \xrightarrow{t_1} v_1 \xrightarrow{t_2} v_2 \ldots v_{n-1} \xrightarrow{t_n} v_n$, such that $\forall\ 1 \leqslant i < n$, $t_i < t_{i+1}$, $\forall\ 1 \leqslant j \leqslant n, v_j \neq v_{j-1}$, i.e., there are no self-loops on the e-temporal path.

The length of an e-temporal path p, denoted by $L(p)$, is the number of temporal edges included in p. Let $S(p) = t_1$ (occurring time of the first edge e_1) and $E(p) = t_n$ (occurring time of the last edge e_n) be the starting time and ending time of p. Let $D(p) = E(p) - S(p) = t_n - t_1$ be the travel time of p. By the above definition, the message is sent immediately but there is a delay between the emission of a message and its reception.

Based on the definition of the temporal graph and e-temporal path, we define three common optimal temporal paths w.r.t. a temporal edge in the following.

Definition 3 (E-shortest temporal path (ESTP). An e-temporal path p from e to v_n is said to be an e-shortest temporal path if there is no other temporal path p' from e to v_n satisfying $L(p') < L(p)$.

Definition 4 (E-earliest temporal path (EETP). An e-temporal path p from e to v_n is recognized as an e-earliest temporal path if there is no other temporal path p' from e to v_n having $E(p') < E(p)$.

Definition 5 (E-fastest temporal path (EFTP). An e-temporal path p from e to v_n is identified as an e-fastest temporal path if there is no other temporal path p' from e to v_n satisfying $D(p') < D(p)$.

Definitions 2 to 5 define the e-temporal path and three e-optimal temporal paths starting from the temporal edge. Accordingly, we can also define v-temporal path (i.e., v-shortest temporal path (VSTP), v-earliest temporal path (VETP) and v-fastest temporal path (VFTP)) starting from the vertex in a similar way. In the following, we do not distinguish v-temporal path and e-temporal path whenever the meaning is clear from the context. Next, we formally define temporal edge closeness and temporal edge betweenness centrality.

Definition 6 (Temporal edge closeness centrality). Given a temporal graph $G = (V, E, T)$, the normalized temporal edge closeness centrality $TECC(u,v,t)$ of any edge $(u,v,t) \in E$ is defined as:

$$TECC(u,v,t) = \frac{1}{|V|-2} \sum_{u \neq f \neq v \in V} \frac{1}{d((u,v,t),f)}. \quad (1)$$

$|V|$ is the number of vertices in G. $d((u,v,t),f)$ is the optimal distance of the e-optimal temporal path from the edge (u,v,t) to f. Specifically, for ESTP, EETP, and EFTP, the optimal distance represents the minimum length, earliest arrival time, and minimum travel time of $p(e = (u,v,t), f)$, respectively. In particular, if (u,v,t) cannot reach f, then $\frac{1}{d((u,v,t),f)} = 0$. Note that, here we consider the temporal analog of the harmonic closeness centrality. since when the temporal graph is weakly connected, this definition helps us deal with the case that two vertices are not connected by any temporal path [19].

Definition 7 (Temporal edge betweenness centrality). Given a temporal graph $G = (V, E, T)$, the normalized temporal edge betweenness centrality $TEBC(u,v,t)$ of any edge $(u,v,t) \in E$ is defined as:

$$TEBC(u,v,t) = \frac{1}{|V|(|V|-1)} \sum_{s \neq f \text{ and } u \neq v \in V} \frac{\sigma_{sf}(u,v,t)}{\sigma_{sf}}. \quad (2)$$

$\sigma_{sf}(u,v,t)$ is the number of v-optimal temporal paths from s to f through (u,v,t); σ_{sf} is the number of v-optimal temporal paths from s to f. In particular, if s cannot reach f via a temporal path, then $\sigma_{sf} = \infty$ and $\frac{\sigma_{sf}(u,v,t)}{\sigma_{sf}} = 0$.

Table 1 Symbols and description

Notation	Description
G, G_c	a temporal graph and the compressed graph
$E_{in}(v)$, $E_{out}(v)$, $E(u,v)$	the set of incoming, outgoing temporal edges of v and the temporal edges from u to v
$T_{in}(v)$, $T_{out}(v)$, $T(u,v)$	the set of distinct timestamps of incoming, outgoing temporal edges of v and the distinct timestamp set of temporal edges from u to v
$p(e, v_n)$	an e-temporal path from e to v_n (see Definition 2)
$L(p), S(p), E(p), D(p)$	the length, the starting time, the ending time, and the travel time of a temporal path p
σ_{su}, σ_{se}	the number of global v-optimal temporal paths from s to u and the number of local v-optimal temporal paths from s whose last edge is e
$P_s(e)$	the list of predecessors of e on local v-optimal temporal paths from s
$\delta_{s.}(e)$	the temporal edge dependency of s on e (see Definition 12)

4 Temporal edge closeness centrality

In this section, we first define the t-label and the t-label dominance relation, and then propose an efficient labeling-based method to compute ECC values of all temporal edges.

According to Definition 6, $TECC(e)$ only needs to compute the optimal distance from e to all the other vertices, instead of listing or counting all the optimal temporal paths from e, which makes the calculation process much easier. It indicates that the optimal distance $d(e,f)$ is obtained once an optimal temporal path from e to f is found. For ECC computation based on ESTP, a feasible way is to perform BFS following the e-temporal path until all the minimum lengths from e to other vertices are determined. While for TECC computation based on EETP and EFTP, the BFS paradigm traverses some

redundant paths and is a waste of time because the shortest temporal path is not necessarily the earliest or fastest temporal path. Quickly locating the optimal temporal path from e and minimizing candidate paths for expansion is crucial. Although not all the prefix subpaths of the optimal temporal paths from e are optimal (as mentioned in Section 1), what we know for certain is, for EETP and EFTP, there must exist an optimal temporal path whose prefix subpaths are also optimal in the set of optimal temporal paths from e, which is guaranteed by the following lemma.

Lemma 1 For EETP and EFTP, there must exist an optimal temporal path whose prefix subpaths are also optimal in the set of optimal temporal paths from e.

Proof First, we prove by contradiction that our statement is true for the case of EETP. Let $EETP(e,f)$ be the set of optimal temporal paths from e to any vertex f. We assume that $\forall p_o(e,f) \in EETP(e,f)$, the prefix subpaths of p_o are not EETPs. Let $p_o(e,f) = u \xrightarrow{t} v \xrightarrow{t_1} v_1 \ldots v_{n-2} \xrightarrow{t_{n-1}} v_{n-1} \xrightarrow{t_n} v_n$, and $p = u \xrightarrow{t} v \xrightarrow{t'_1} v_1 \ldots v_{n-2} \xrightarrow{t'_{n-1}} v_{n-1}$ is the temporal path from e to v_{n-1} whose prefix subpaths are EETPs. Then we have $t'_{n-1} < t_{n-1} < t_n$, and thus a new temporal path $p'(e,f) = u \xrightarrow{t} v \xrightarrow{t'_1} v_1 \ldots v_{n-2} \xrightarrow{t'_{n-1}} v_{n-1} \xrightarrow{t_n} v_n$ extended by p is an EETP from e to f whose prefix subpaths are EETPs, it must be contained in $EETP(e,f)$ by Definition 4, which is a contradiction. Similarly, we can prove that the above statement is true for the case of EFTP. The proof completes. □

Based on Lemma 1, we propose a labeling-based method. The basic idea is to define and exploit refined time-related labels to seek out the optimal temporal path from an edge e whose prefix subpaths are also optimal, and use it for the extension to efficiently attain the optimal distance from e to all the other vertices. Additionally, the temporal edges from which the sets of optimal temporal paths are equal are aggregated and calculated once to reduce the number of computations.

Definition 8 (t-label). A t-label of edge e is identified by $l_e = (v, a_t)$ tuple, where a_t is the ending time of the temporal path from e to v.

Maintaining and extending all the t-labels of edges is not necessary because not all the t-labels can generate the optimal temporal paths whose prefix subpaths are also optimal, hence we define the t-label dominance relation to screen out these t-labels, and thus reduce the invalid or redundant extensions.

Definition 9 (t-label dominance relation). A t-label $l'_e = (v, a'_t)$ is dominated by another one $l_e = (v, a_t)$ if $a'_t > a_t$.

The dominated t-labels l'_e can be safely pruned because the temporal paths extended by l'_e can also be expanded by l_e. In Fig. 2(a), there are 2 e-temporal paths from $e = (a,b,1)$ to f, i.e., $p_1 = a \xrightarrow{1} b \xrightarrow{3} f$ and $p_2 = a \xrightarrow{1} b \xrightarrow{5} f$, which generate $l_1 = (f,3)$ and $l_2 = (f,5)$ by Definition 8. According to Definition 9, l_2 is dominated by l_1 as $3 < 5$.

In this paper, we aim at computing ECC values of all temporal edges. By our observation, the sets of optimal temporal paths from some temporal edges are equal, and thus these edges are placed in the same group for uniform processing.

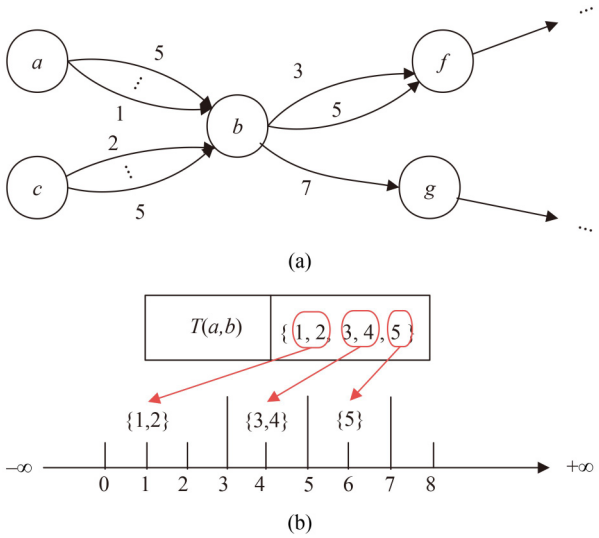


Fig. 2 Example of a temporal graph and the time segmentation strategy. (a) A temporal graph G; (b) time segmentation result of $T(a, b)$

Observation 1 For a temporal edge $e = (u,v,t)$, u is the starting vertex, and v is the ending vertex. The sets of optimal temporal paths from any temporal edge with the same ending vertex and the same timestamp to all the other vertices are equal.

The reason why Observation 1 is valid is that the set of vertices, as well as the corresponding ending time, reached from the temporal edges that have the same ending vertex and the same timestamp, are completely consistent.

Before presenting Observation 2, we first provide a time segmentation strategy.

Time segmentation strategy $\forall v \in V$, The time segmentation strategy is to divide the timestamps of $v's$ incoming temporal edges from u into several subsets by the timestamps of $v's$ outgoing edges. Formally, let $T(u,v) = \{t_1, t_2, ..., t_m\}$ and $T_{out}(v) = \{t'_1, t'_2, ..., t'_k\}$. $T_{out}(v)$ divides $T(u,v)$ into $k+1$ subsets $(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{k+1})$ at most, such that $\mathcal{T}_1 = \{t_i \mid t_i \in T(u,v), t_i \in (-\infty, t'_1)\}$, $\mathcal{T}_j = \{t_i \mid t_i \in T(u,v), t_i \in [t'_{j-1}, t'_j)\}$ $(1 < j \leqslant k)$, and $\mathcal{T}_{k+1} = \{t_i \mid t_i \in T(u,v), t_i \in [t'_k, +\infty)\}$.

Observation 2 By the time segmentation strategy, the temporal edges from u to v are divided into corresponding subsets. The sets of optimal temporal paths from the temporal edges that fall into the same subset to any vertex are equal.

Example 2 Figure 2(a) shows a temporal graph, where the edges are labeled with the timestamps of their existence. There are 5 and 4 temporal edges from a to b and c to b, respectively, and the timestamp difference is 1. The outgoing temporal edges of f and g are omitted. According to Observation 1, $\forall t \in \{2,3,4,5\}$, the sets of optimal temporal paths from (a,b,t) and (c,b,t) to other vertices are equal. Figure 2(b) shows how the timestamps of b's incoming temporal edges from a are segmented by the timestamps of b's

Algorithm 1 TECC algorithm

Input: A temporal graph $G = (V, E, T)$, the number of threads #*threadnum*
Output: *TECC* values of all edges $(u, v, t) \in E$

1: create #*threadnum* threads; $\forall (u, v, t) \in E$, $TECC(u, v, t) \leftarrow 0$
2: **for** incoming edges (u, v, t_v) of v with same t_v **do**
3: aggregate these edges into a *group* // by Observation 1
4: **foreach** u and all the incoming edges of v from u **do**
5: divide the timestamps associated with $E(u, v)$ by time segment strategy
6: aggregate the edges falling into the same subset into the same *group* // by Observation 2
7: **foreach** *group* delegated to thread i ($1 < i \leq$ #*threadnum*) **do**
8: randomly select an edge $e = (u, v, t)$ from the *group*
9: Compute(d, e)
10: **foreach** edge $e' = (x, v, t')$ in the *group* **do**
11: $TECC(e') = \frac{1}{|V|-2} \sum_{x \neq f \neq v \in V} \frac{1}{d(e,f)+t-t'}$ // by Definition 6
12: return $TECC$
13: **Procedure** Compute$(d, e = (u, v, t))$
14: Initialize $Q \leftarrow \emptyset$; $F \leftarrow \emptyset$; $\forall f \in V$, $d(e, f) \leftarrow \infty$
15: $l = (v, t)$, $Q.insert(l)$
16: **while** Q *not empty* **do**
17: $l = (f, t_f) \leftarrow Q.pop()$
18: $d(e, f) = t_f - t$, $F = F \cup f$
19: **foreach** $(f, w, t_w) \in E_{out}(f)$ **do**
20: **if** $t_f < t_w$ *and* $w \notin F$ **then**
21: $l' = (w, t_w)$; $Q.insert(l')$
22: remove dominated t-labels from Q // by Definition 9

outgoing edges according to the proposed time segmentation strategy. In Fig. 2(a), $T(a,b) = \{1,2,3,4,5\}$, and $T_{out}(b) = \{3,5,7\}$. Then $T(a,b)$ is split into 3 subsets (i.e., $\{1,2\}$, $\{3,4\}$, and $\{5\}$) by $T_{out}(b)\}$, as depicted in Fig. 2(b). Correspondingly, $E(a,b)$ is divided into 3 groups, i.e., $\{(a,b,1),(a,b,2)\}$, $\{(a,b,3),(a,b,4)\}$, and $\{(a,b,5)\}$.

Based on the above definitions and observations, we present the TECC algorithm based on EFTP (TECC based on EETP will be clarified later). Considering that the ECC calculations for each edge are relatively independent, TECC is a multi-threading implementation using OpenMP for higher parallelism. OpenMP delegates a user-specified number of threads to compute ECC values in parallel.

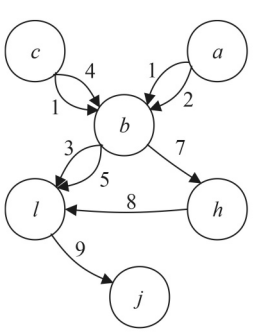
The pseudo-code of TECC is shown in Algorithm 1. TECC takes as inputs a temporal graph G and the user-specified number of threads #*threadnum*, and it outputs $TECC$ values of all edges. First of all, TECC aggregates the edges with the same ending vertex and the same timestamp into a group according to Observation 1 (lines 2−3); and divides the timestamps associated with the incoming edges into multiple subsets by the time segment strategy (lines 4−5). The edges in the same subset fall into the same group by Observation 2 (line 6). In the sequel, TECC randomly chooses an edge $e = (u, v, t)$ from each group, and delegates a thread to invoke the Compute function to calculate the minimum travel time $d(e, f)$ from e to all the other vertices f (lines 7−9). By Observation 2, the sets of optimal temporal paths from the temporal edges that fall into the same subset to any vertex are equal, hence the ending time of the temporal paths from the edges in the same group to all the other vertices f is equal, which is $d(e, f) + t$. Based on this conclusion and Definition 6, the ECC values of all edges in the same group are calculated (lines 10−11).

Compute(d, e) exploits t-labels to iteratively expand EFTP whose prefix subpaths are also EFTP rather than all the temporal paths, and calculates the minimum travel time from e to all the other vertices. First, Compute initializes a priority queue Q, a set F and optimal distances $d(e, f)$ from e to each f in V (line 14). Q maintains t-labels sorted in non-descending order of their timestamps. F maintains vertices, to which the minimum travel time from e is determined. $d(e, f)$ stores the minimum travel time from e to f. In the beginning, a t-label $l = (v, t)$ generated by the edge $e = (u, v, t)$ is inserted into Q (line 15). Next, a while loop is performed. Compute determines $d(e, f)$ and extends the optimal temporal path with the t-labels until Q is empty or the minimum travel time from e to all the other vertices is calculated (lines 16−22). In each iteration, Compute first pops the t-label $l = (f, t_f)$ with the smallest timestamp from Q, then the minimum travel time $d(e, f)$ is calculated, and thus f is added to F (lines 17−18). Next, for each outgoing edge (f, w, t_w) of f, Compute creates the t-label (w, t_w) and inserts it into Q (lines 19−21). Thereafter, Compute removes dominated t-labels from Q by Definition 9 to shrink the search space (line 22).

Next, we show how TECC works with the example below.

Example 3 Consider a temporal graph shown in Fig. 3(a). First, the temporal edges are aggregated into 6 groups by Observations 1 and 2 (lines 2−6). Thus, the number of computations is reduced from $|E|$ (=9) to 6. For instance, edges from a and c to b are divided into 2 sets, (i.e., $S_1 = \{(a,b,1),(a,b,2),(c,b,1)\}$, $S_2 = \{(c,b,4)\}$). Take the calculation of S_1 as a representative. TECC randomly selects an edge $(a,b,1)$ from S_1 and then invokes Compute function to iteratively expand the optimal temporal paths and calculate minimum travel time $d((a,b,1), f)$ from $(a,b,1)$ to all the other vertices f (lines 13−22). Initially, $e = (a,b,1), Q = \{(b,1)\}$, then a while-loop is executed.

Loop 1: $(b,1)$ is popped, which indicates the minimum travel time from $(a,b,1)$ to b is confirmed. Hence, $d((a,b,1),b) = 0$, $F = \{b\}$. Then new t-labels $(l,3)$, $(l,5)$, $(h,7)$ are generated by $(b,1)$, but only the un-dominated ones $(l,3)$



(a)

f	$d((a,b,1),f)$
l	$d((a,b,1),l)=2$
h	$d((a,b,1),h)=6$
j	$d((a,b,1),j)=8$

(b)

Fig. 3 Example of TECC calculation. (a) A temporal graph G; (b) result of $d((a, b, 1), f)$

and $(h,7)$ are pushed into Q. Thus $Q=\{(l,3),(h,7)\}$.

Loop 2: $(l,3)$ is popped, then $d((a,b,1),l)=2$, $F=\{b,l\}$. the un-dominated t-label $(j,9)$ is pushed into Q.

The while loop stops until the minimum travel time from $(a,b,1)$ to all the other vertices is confirmed, as illustrated in Fig. 3. Then, $TECC$ values of all the edges in S_1 can be computed (lines 10−11 of Algorithm 1). Finally, $TECC(a,b,1)=\frac{19}{24}$, $TECC(a,b,2)=\frac{47}{35}$, $TECC(c,b,1)=\frac{19}{24}$.

Discussion. It is not difficult to find that an EETP from $e=(u,v,t)$ to any other vertex f is also an EFTP, because the earliest ending time E_p of an EETP leads to the minimum travel time E_p-t. Hence the idea of calculating $TECC(e)$ based on EETP is the same as that of calculating $TECC(e)$ based on EFTP. The minor differences are that, for EETP, Compute function calculates the earliest ending time $d(e,f)$ from e to f (line 19 changes to $d(e,f)=t_f$); and $TECC(e)$ is computed based on EETP (line 11 changes to $TECC(e')=\frac{1}{|V|-2}\sum_{x\neq f\neq v\in V}\frac{1}{d(e,f)}$)).

Time and space complexities of TECC**.** Compute is the main function in TECC. For Compute, there are at most $|V|$ t-labels in Q because, by Definition 9, each vertex v corresponds to a unique t-label with the minimum timestamp a_t. Hence inserting or popping t-labels from Q needs $O(log|V|)$ cost. Thus the cost of Compute is $O(|V|+|E||log|V|)$. Correspondingly, the time complexity of TECC is $O(\frac{|group|(|V|+|E|log|V|)}{\#threadnum})$, where $|group|$ is the number of groups divided by Observations 1 and 2.

TECC algorithm maintains optimal distances $d(e,f)$ from e to each vertex f, a priority queue Q and a set F for each thread, which needs $O(|V|)$ space. The adjacency list of the input temporal graph requires $O(|V|+|E|)$ space. Thus, the space complexity of TECC is $O(\#threadnum(|V|+|E|))$.

5 Temporal edge betweenness centrality

Calculating the EBC values of all edges is much more difficult than ECC because it first needs to enumerate all v-optimal temporal paths between pairs of vertices, and then computes the contributions of the optimal temporal paths containing the edge by Definition 7. In contrast, ECC no longer needs to find other optimal temporal paths once an optimal temporal path is found.

For EBC computation, the traditional greedy-based method for finding the shortest paths in general graphs is inapplicable because prefix subpaths of an optimal temporal path may not be optimal, hence intuitively all the potential e-temporal paths need to be maintained as candidates and be extended, resulting in prohibitively time and space overhead. To avoid enumerating all potential temporal paths, we extend the concept of t-label and t-label dominance relation defined in Definitions 8 and 9 to v-label and v-label dominance relation, respectively, propose a temporal edge dependency formula and a labeling-based forward-backward scanning strategy to iteratively accumulate the contributions of the subpaths to their containing edges.

Definition 10 (v-label). A v-label of a source vertex s is defined as $l_s=(e,s_t)$, where s_t is the starting time of the temporal path from s to v where the last edge is $e=(u,v,t_v)$.

Definition 11 (v-label dominance relation). A v-label $l'_s=((u,v,t'_v),s'_t)$ dominates another one $l_s=((u,v,t_v),s_t)$ if $t'_v=t_v$ and $s'_t>s_t$.

To find all the optimal temporal paths without any omission, Definition 11 is more strict than Definition 9. The v-labels $l'_s=((u,v,t'_v),s'_t)$ can be safely filtered out for EBC computation based on EFTP only if there exists another v-label $l_s=((u,v,t_v),s_t)$ having $t'_v=t_v$ and $s'_t>s_t$. Next, we introduce temporal edge dependency and derive a temporal edge dependency formula.

Definition 12 (Temporal edge dependency). The temporal edge dependency of s on $e=(u,v,t_v)$ is defined as $\delta_{s.}(e)=\sum_{s\neq f\wedge u\neq v\in V}\delta_{sf}(e)=\sum_{s\neq f\wedge u\neq v\in V}\frac{\sigma_{sf}(e)}{\sigma_{sf}}$, where $\sigma_{sf}(e)$ is the number of v-optimal temporal paths from s to f via e. σ_{sf} is the number of optimal temporal paths from s to f.

Lemma 2 The temporal edge dependency $\delta_{s.}(e=(u,v,t_v))$ of s complies with the following:

$$\begin{aligned}\delta_{s.}(u,v,t_v)&=\sum_{s\neq f\wedge u\neq v\in V}\delta_{sf}(e)=\sum_{s\neq f\wedge u\neq v\in V}\frac{\sigma_{sf}(e)}{\sigma_{sf}}\\&=\sum_{e'=(v,w,t_w):e\in P_s(e')}\left(\frac{\sigma_{se}\cdot Flag(e)}{\sigma_{sv}}+\frac{\sigma_{se}}{\sigma_{se'}}\cdot\delta_{s.}(e')\right).\end{aligned}\tag{3}$$

Here, σ_{se} is the number of local v-optimal temporal paths from s to v in which the last edge is $e(=(u,v,t))$; $P_s(e')$ is the list of predecessors of e' on local v-optimal temporal paths from s. It is noteworthy that the local v-optimal temporal path from s to v where the last edge is $e(=(u,v,t))$ does not necessarily be the global optimal temporal path from s to v. Thus, we set $Flag(e)$ to mark whether e is the last edge of the global optimal temporal path from s to v. $Flag(e)=1$ means that e is the last edge of the optimal temporal path, otherwise, $Flag(e)=0$.

Proof If s cannot reach f via a temporal path, then $\sigma_{sf}=\infty$, so in the following proof, we give priority to discussing the case where $\sigma_{sf}\neq\infty$.

$$\begin{aligned}\delta_{s.}(e)&=\sum_{s\neq f\wedge u\neq v\in V}\delta_{sf}(e)\\&=\sum_{s\neq f\wedge u\neq v\in V}\sum_{e':e\in P_s(e'))}\delta_{sf}(e\to e')\\&=\sum_{e':e\in P_s(e')}\sum_{s\neq f\wedge u\neq v\in V}\frac{\sigma_{sf}(e\to e')}{\sigma_{sf}}.\end{aligned}\tag{4}$$

$\delta_{sf}(e\to e')$ is the temporal edges dependency that pass e to e'. $\sigma_{sf}(e\to e')$ is the number of optimal temporal paths from s to f via consecutive edges e and e'. There are two circumstances:

(i) $v=f$, e.g., $s\cdots\to e$, that is, e is the last edge of the v-optimal temporal path:

$$\delta_{sf}(e)=\frac{\sigma_{se}}{\sigma_{sf}}.\tag{5}$$

(ii) $v\neq f$, e.g., $s\cdots\to e\to e'\to\cdots\to f$. The v-optimal

temporal paths from s to f are through e and e':

$$\begin{aligned}\delta_{sf}(e \to e') &= \frac{\sigma_{se} \cdot \sigma_{e'f}}{\sigma_{sf}} = \frac{\sigma_{se} \cdot \sigma_{e'f} \cdot \sigma_{se'}}{\sigma_{sf} \cdot \sigma_{se'}} \\ &= \frac{\sigma_{se'} \cdot \sigma_{e'f}}{\sigma_{sf}} \cdot \frac{\sigma_{se}}{\sigma_{se'}} \\ &= \delta_{sf}(e') \cdot \frac{\sigma_{se}}{\sigma_{se'}}. \end{aligned} \tag{6}$$

However, it is still necessary to consider the case that the local v-optimal temporal path from s to v through e is not the global v-optimal temporal path from s to v, so we multiply Eq. (5) by $Flag(e)$ to judge case (i). In case(ii), if there is no v-optimal temporal path from s to f through e, the contribution of optimal temporal paths will not be propagated to e through the predecessor list of f, so case (ii) is not affected, hence:

$$\delta_{s.}(e) = \sum_{e':e\in P_s(e')} \left(\frac{\sigma_{se} \cdot Flag(e)}{\sigma_{sf}} + \frac{\sigma_{se}}{\sigma_{se'}} \cdot \delta_{s.}(e') \right). \tag{7}$$

The proof is complete. □

Based on the v-label, v-label dominance relation, and the temporal edge dependency formula, we propose TEBC algorithm based on EFTP, which is outlined in Algorithm 2. TEBC is also a multi-threading implementation using OpenMP. In summary, TEBC mainly adopts a labeling-based forward-backward scanning strategy in function Compute to iteratively accumulate the contributions of the subpaths to their containing edges. In forward scanning, TEBC generates the v-labels and then progressively computes optimal temporal paths (lines 13−23). In the backward scanning, starting from the optimal temporal paths with the maximum length, TEBC iteratively accumulates the contribution of the subpaths to their containing edges by Lemma 2 (lines 24−30).

Specifically, TEBC takes as inputs a temporal graph G and the user-specified number of threads #*threadnum*, and it outputs $TEBC$ values of all edges. TEBC traversals all the source vertices s and delegates a thread to invoke the Compute function to calculate the $TEBC$ value of all the edges (line 2−3). Compute first performs initialization (line 6−10), where $d(s,u)$ and $d(s,e)$ are the minimum travel time of the VFTP from s to u and the maximum length of the local optimal temporal path from s whose last edge is e, respectively; σ_{su} and σ_{se} are the number of global v-fastest temporal paths from s to u and the number of local v-fastest temporal paths from s whose last edge is e, respectively; $P_s(e)$ maintains the predecessors of e on local v-optimal temporal paths from s; $Flag(e)$ indicates whether e is the last edge on the v-fastest temporal path from s; Q and S are priority queues. Q maintains v-labels $(e=(u,v,t_v),t)$ sorted in the non-descending order of $t_v - t$. In the beginning, v-labels generated by the outgoing edges of s are inserted into Q (lines 11−12). S records edges e that are the last edges of optimal temporal paths from s sorted in non-ascending order of $d(s,e)$.

Next, Compute proceeds with forward scanning with a while loop (lines 13−23). In each loop, Compute first pops the v-label $l=(e=(u,v,t_v),t)$ with the minimum $t_v - t$ from Q, then updates $d(s,v)$, σ_{sv}, $Flag(e)$ and S (lines 14−17). After

Algorithm 2 TEBC algorithm

Input: A temporal graph $G=(V,E,T)$, number of threads #*threadnum*
Output: $TEBC$ of all edges $e=(u,v,t_v)\in E$

1: create #*threadnum* thread; $\forall(u,v,t_v)\in E$, $TEBC(u,v,t_v)\leftarrow 0$
2: **foreach** $s\in V$ **do**
3: thread i ($1\le i\le$ #*threadnum*) executes Compute($TEBC$, s)
4: **return** $TEBC$
5: **Procedure** Compute($TEBC$, s)
6: **foreach** $u\in V$ **do**
7: $d(s,u)\leftarrow -1$; $\sigma_{su}\leftarrow 0$
8: **foreach** $e=(u,v,t_v)\in E$ **do**
9: $d(s,e)\leftarrow -1$; σ_{se}, $\delta_{s.}(e)$, $Flag(e)\leftarrow 0$; $P_s(e)\leftarrow\emptyset$;
10: Initialize $Q,S\leftarrow\emptyset$; $d(s,s)=0$
11: **foreach** $e=(s,v,t_v)\in E_{out}(s)$ **do**
12: $l_s=(e,t_v)$; $Q.insert(l_s)$; $\sigma_{se}=1$; $d(s,e)=0$
13: **while** Q *not empty* **do**
14: $l_s=(e=(u,v,t_v),t)\leftarrow pop(Q)$
15: **if** $d(s,v)=-1$ or $d(s,v)=t_v-t$ **then**
16: $d(s,v)=t_v-t$; $\sigma_{sv}+=\sigma_{se}$;
17: $Flag(e)\leftarrow 1$; $S.insert(e)$
18: **foreach** $e'=(v,w,t_w)\in E_{out}(v)$ *with* $t_v<t_w$ **do**
19: $l'_s=(e',t)$; $Q.insert(l'_s)$;
20: remove dominated v-labels from Q
21: **if** l' *is not dominated* **then**
22: $\sigma_{se'}+=\sigma_{se}$; $P_s(e')\leftarrow P_s(e')\cup e$;
23: $d(s,e')=max(d(s,e)+1,d(s,e'))$
24: **while** S *not empty* **do**
25: $e'=(v,w,t_w)\leftarrow$ pop(S);
26: **if** $Flag(e')=1$ **then**
27: $\delta_{s.}(e')+=\frac{\sigma_{se'}}{\sigma_{sw}}$; $TEBC(e)+=\frac{\sigma_{se'}}{\sigma_{sw}}\cdot\frac{1}{|V|(|V|-1)}$
28: **foreach** $e=(u,v,t)\in P_s(e')$ **do**
29: $\delta_{s.}(e)+=\frac{\sigma_{se}}{\sigma_{se'}}\cdot\delta_{s.}(e')$
30: $TEBC(e)+=\frac{\sigma_{se}}{\sigma_{se'}}\cdot\delta_{s.}(e')\cdot\frac{1}{|V|(|V|-1)}$

that, for each outgoing edge $e'=(v,w,t_w)$ of v, Compute creates the v-label $l'=(e',t)$ and inserts it into Q (lines 18−19). Thereafter, dominated v-labels are removed from Q by Definition 11. $\sigma_s(e')$, $P_s(e')$, and $d(s,e')$ are updated if l' is not dominated (line 21−23). In the sequel, Compute performs backward scanning (lines 24−30). The contributions of FTPs from s are iteratively accumulated to their containing edges by Lemma 2 until Q is empty.

Discussion. TEBC algorithm based on VETP is similar to VFTP, but there are two differences, listed as follows. (i) v-label dominance relation is invalid for VETP as the dominated v-labels can also generate VETPs (line 20 of Algorithm 2 should be deleted for VETP); and (ii) for VETP, Compute function calculates the earliest ending time $d(s,v)$ from source vertex s to v (lines 15 and 16, $d(s,v)=t_v$). Correspondingly, Q maintains v-labels $(e=(u,v,t_v),t)$ sorted in the non-descending order of t_v instead of t_v-t. As for VSTP, in the forward scanning, we only need to perform the temporal BFS paradigm, which naturally ensures that local v-shortest temporal paths found are length-ordered. Hence, a priority queue Q is enough for VSTP.

Time and space complexities of TEBC. Let γ_e be the number of times the edge e is visited, and θ_e be the number of adjacent edges that are temporally reached by e. TEBC invokes Compute function to compute EBC values. For Compute, lines 6−12 of TEBC perform initialization,

which takes $O(|V|+|E|)$ time. Lines 13−22 take $O(log|E|\sum_{e\in E}\gamma_e\theta_e)$; lines 23−29 are conducted in $O(\sum_{e\in E}|P_{s.}(e)|)$ time. Thus the time complexity of TEBC is $O(\frac{|V|(|V|+|E|+log|E|\sum_{e\in E}\gamma_e\theta_e+\sum_{e\in E}|P_{s.}(e)|)}{\#threadnum})$. The time complexity of the VETP-based algorithm is the same as that of TEBC. The time complexity of the VSTP-based algorithm is $O(\frac{|V|(|V|+|E|+|E|\hat{E}_{out}+\sum_{e\in E}|Ps.(e)|)}{\#threadnum})$, where $\hat{E}_{out}$ is the maximum number of out-going edges of any vertex.

TEBC algorithm stores $TEBC$ values in $O(|E|)$ space. In each thread, $d(s,v)$ and σ_{sv} are stored in $O(|V|)$ space. $d(s,e)$, σ_{se}, $\delta_{s.}(e)$, $Flag(e)$, Q and S need $O(|E|)$ space. $P_s(e)$ needs $O(|E|\hat{E}_{in})$ space, where $\hat{E}_{in}$ is the maximum number of incoming edges of any vertex. Thus, the space complexity of TEBC is $O(\#threadnum(|V|+|E|+|E|\hat{E}_{in})+|E|)$. The space complexities of Algorithms based on VETP and VSTP are the same as that of TEBC.

Optimization. We observe that throughout the EBC computation process, some temporal edges in G not only play the same roles but also have closely related EBC values. These edges can be compressed and further jointly calculated to reduce memory consumption and improve efficiency.

Temporal edge compression method. Let $E_{out}(u,v,t_v)=\{(v,w,t_v)\,|\,(v,w,t_v)\in E_{out}(v),t_w>t_v\}$ and $E_{in}(u,v,t_v)=\{(x,u,t_u)\,|\,(x,u,t_u)\in E_{in}(u),t_u<t_v\}$ be the sets of edge (u,v,t)'s outgoing and incoming temporal edges, respectively. The edges e_1, e_2, …, e_n can be compressed if $E_{in}(e_1)=E_{in}(e_2)=\cdots=E_{in}(e_n)$ and $E_{out}(e_1)=E_{out}(e_2)=\cdots=E_{out}(e_n)$. For example in Fig. 4(a), $E_{in}(b,c,2)=E_{in}(b,c,3)=\{(a,b,1)\}$ and $E_{out}(b,c,2)=E_{out}(b,c,3)=\{(c,g,7),(c,g,8),(c,g,10)\}$, hence $(b,c,2)$ and $(b,c,3)$ can be compressed into $(b,c,\{2,3\})$. Similarly, $(c,g,7)$, $(c,g,8)$, and $(c,g,10)$ are compressed into $(c,g,\{7,8,10\})$; $(a,h,4)$ and $(a,h,5)$ are compressed into $\{a,h,\{4,5\}\}$; $(h,g,7)$ and $(h,g,8)$ are compressed into $\{h,g,\{7,8\}\}$; $(g,f,12)$ and $(g,f,13)$ are compressed into $\{g,f,\{12,13\}\}$, as shown in Fig. 4(b).

The pseudo-code of the Edge Compression Optimization (ECO) method is shown in Algorithm 3. ECO takes as inputs a temporal graph G, and it outputs the compressed graph $G_c=(V_c,E_c,T_c)$. For each vertex pair (u,v) and temporal edges in $E(u,v)$ that are not compressed, if the temporal edges have equal E_{in} and E_{out}, then they are compressed (lines 2−6).

Time and space complexities of ECO. The time complexity of ECO is $O(\sum_{u,v\in V}\sum_{(u,v,t_v)\in E(u,v)}|E(u,v)|(|E_{out}(u,v,t_v)|+|E_{in}(u,v,t_v)|))$, and the space overhead of ECO

Algorithm 3 ECO algorithm

Input: temporal graph $G=(V,E,T)$
Output: compressed graph $G_c=(V_c,E_c,T_c)$
1: $V_c=V$; $E_c=\emptyset$; $T_c=\emptyset$
2: **foreach** vertex pair (u,v), $u,v\in V$ and $e=(u,v,t_v)\in E(u,v)$ that is not compressed **do**
  // $E(u,v)$ is sorted by timestamps in non-descending order
3: $\tau=t_v$
4: **foreach** other $e'=(u,v,t'_v)\in E(u,v)$ that is not compressed **do**
5: **if** $E_{out}(e)=E_{out}(e')$ and $E_{in}(e)=E_{in}(e')$ **then**
6: $\tau=\tau\cup t'_v$
7: $E_c=E_c\cup(u,v,\tau)$; $T_c=T_c\cup\tau$
8: **return** G_c

is $O(|V|+|E|+|V_c|+|E_c|)$ since the original and compressed temporal graphs need to be stored.

Based on G_c, the temporal edge dependency formula presented in Lemma 2 should be revised below.

Lemma 3 When the original temporal edge $e=(u,v,t_v)\in G$ is compressed into $e_c=(u,v,\tau_v)\in G_c$, the temporal edge dependency $\delta_{s.}(e_c)$ is transformed into:

$$\delta_{s.}(e_c)=\begin{cases} based\ on\ VSTP: \\ \sum\limits_{e'_c=(v,w,\tau_w):e_c\in P_s(e'_c)}\left(\frac{\sigma_{se_c}}{\sigma_{sv}}\cdot Flag(e_c)+\frac{\sigma_{se_c}\cdot|\tau_w|}{\sigma_{se'_c}}\cdot\delta_{s.}(e'_c)\right), \\ based\ on\ VFTP\ and\ VETP: \\ \sum\limits_{e'_c=(v,w,\tau_w):e_c\in P_s(e'_c)}\left(\frac{\sigma_{se_c}}{\sigma_{sv}\cdot|\tau_v|}\cdot Flag(e_c)+\frac{\sigma_{se_c}\cdot|\tau_w|}{\sigma_{se'_c}}\cdot\delta_{s.}(e'_c)\right), \end{cases} \tag{8}$$

where

$$\sigma_{se_c}=\begin{cases}|\tau_v|, & e_c\in E_{out}(s),\\ \sum\limits_{\forall e''_c\in P_s(e_c)}\sigma_{se''_c}\cdot|\tau_v|, & \text{otherwise.}\end{cases} \tag{9}$$

$|\tau_v|$ is the number of compressed edges by e_c.

Proof Traversing the compressed edge $e_c=(u,v,\tau_v)$ results in $|\tau_v|$ paths, hence $\sigma_{se_c}=\sum_{\forall e''_c\in P_s(e_c)}\sigma_{se''_c}\cdot|\tau_v|$. By Lemma 2, there are two cases:

Case (i): e_c is the last edge of the optimal temporal path from s to f, then for VSTP, it still has:

$$\delta_{sf}(e_c)=\frac{\sigma_{se_c}\cdot Flag(e_c)}{\sigma_{sf}}. \tag{10}$$

While for VFTP and VETP,

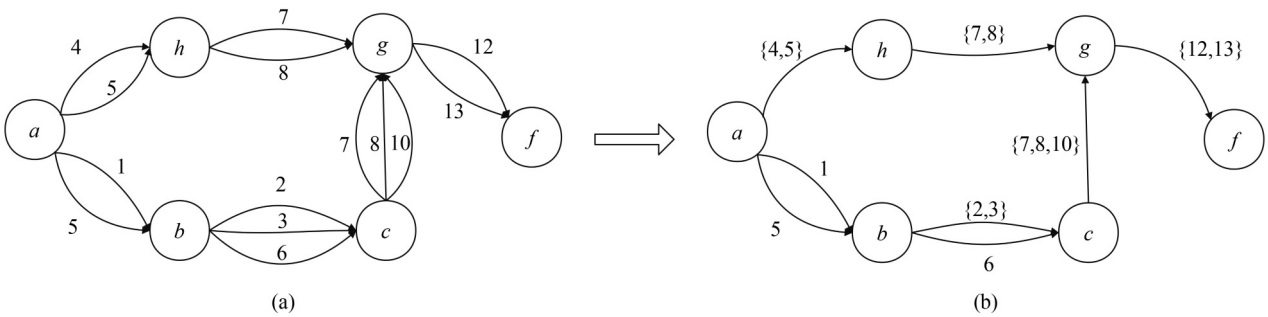


Fig. 4 Examples of a temporal graph and the compressed graph. (a) A temporal graph; (b) a compressed graph

$$\delta_{sf}(e_c) = \frac{\sigma_{se_c} \cdot Flag(e_c)}{\sigma_{sf} \cdot |\tau_v|}. \quad (11)$$

This is because, for VSTP, all the original temporal edges compressed by e_c are included in the optimal temporal path from s to f. In contrast, for VFTP and VETP, among all the original temporal edges compressed by e_c, only the one with the minimum timestamp is actually the last edge of the optimal temporal path from s to f.

Case (ii): The optimal temporal paths from s to f are through e_c and e'_c, then whether VSTP or VFTP, VETP, it has:

$$\delta_{sf}(e_c \to e'_c) = \frac{\sigma_{se_c} \cdot \sigma_{e'_c f} \cdot |\tau_w|}{\sigma_{sf}} = \delta_{sf}(e'_c) \cdot \frac{\sigma_{se_c} \cdot |\tau_w|}{\sigma_{se'_c}}. \quad (12)$$

Combining cases (i) and (ii), the lemma is proved. □

Lemma 3 enables ETBC to efficiently compute the EBC values $TEBC(e_c)$ of the compressed edges $e_c = (u, v, \tau_v)$. Next, we explain how to compute the EBC values of the original edges compressed by e_c.

For EBC computation based on VSTP, obviously, $TEBC(e_c)$ is evenly assigned to the original edges compressed by e_c, i.e., for any e that is compressed by e_c, $TEBC(e) = \frac{TEBC(e_c)}{|\tau_v|}$. However, for EBC computation based on VETP and VFTP, it is a little complicated. $\delta_{s.}(e)$ and $TEBC(e)$ are obtained by Lemma 4.

Lemma 4 For EBC computation based on VETP and VFTP, by Lemma 3, we have $\delta_{s.}(e_c) = \sum_{e'_c=(v,w,\tau_w):e_c \in P_s(e'_c)} (\mathcal{P}_{e_c} + \mathcal{P}_{e'_c})$, where

$$\mathcal{P}_{e_c} = \frac{\sigma_{se_c}}{\sigma_{sv} \cdot |\tau_v|} \cdot Flag(e_c), \quad (13)$$

$$\mathcal{P}_{e'_c} = \frac{\sigma_{se_c} \cdot |\tau_w|}{\sigma_{se'_c}} \cdot \delta_{s.}(e'_c). \quad (14)$$

If e has the minimum timestamp among all the original temporal edges compressed by e_c, then:

$$\delta_{s.}(e) = \sum_{e'_c=(v,w,\tau_w):e_c \in P_s(e'_c)} (\mathcal{P}_{e_c} + \frac{\mathcal{P}_{e'_c}}{|\tau_v|}). \quad (15)$$

Otherwise:

$$\delta_{s.}(e) = \sum_{e'_c=(v,w,\tau_w):e_c \in P_s(e'_c)} \frac{\mathcal{P}_{e'_c}}{|\tau_v|}. \quad (16)$$

By Definition 7 and Definition 12,

$$TEBC(e) = \begin{cases} e \text{ has the minimum timestamp}: \\ \frac{1}{|V|(|V|-1)} \sum_{s \in V} \sum_{e'_c=(v,w,\tau_w):e_c \in P_s(e'_c)} \left(\mathcal{P}_{e_c} + \frac{\mathcal{P}_{e'_c}}{|\tau_v|}\right), \\ otherwise: \\ \frac{1}{|V|(|V|-1)} \sum_{s \in V} \sum_{e'_c=(v,w,\tau_w):e_c \in P_s(e'_c)} \frac{\mathcal{P}_{e'_c}}{|\tau_v|}. \end{cases} \quad (17)$$

Similar to Lemma 3, Lemma 4 still needs to think about two situations. The contributions of cases (i) and (ii) are denoted by $\mathcal{P}_{e_c}$ and $\mathcal{P}_{e'_c}$, respectively. For VETP and VFTP, among all the edges compressed by e_c, only the edge e with the minimum timestamp is the real last edge of the VETP and VFTP in G. Hence, $\mathcal{P}_{e_c}$ contributes to $\delta_{s.}(e)$ instead of other edges compressed by e_c. As for $\mathcal{P}_{e'_c}$, it is evenly assigned to the edges compressed by e_c.

Example 4 Take Fig. 4 as an example. In the forward scanning from vertex a based on VFTP, the information recorded by TEBC is illustrated in Fig. 5. In the backward scanning stage, first it is not difficult to calculate $\delta_{a.}(g, f, \{12, 13\}) = 1$. Then TEBC computes $\delta_{a.}(c, g, \{7, 8\})$ and $\delta_{a.}(h, g, \{7, 8\})$ by Lemmas 3 and 4.

$$\delta_{a.}(c, g, \{7, 8, 10\}) = \frac{\sigma_{a(c,g,\{7,8,10\})} \cdot Flag(c, g, \{7, 8, 10\})}{\sigma_{ag} \cdot |\{7, 8, 10\}|} + \frac{\sigma_{a(c,g,\{7,8,10\})} \cdot |\{12, 13\}|}{\sigma_{a(g,f,\{12,13\})}} \cdot \delta_{a(g,f,\{12,13\})} = \frac{3}{5} + \frac{1}{2} = \frac{11}{10};$$

$$\delta_{a.}(c, g, 7) = \frac{1}{2} + \frac{3}{5} \times \frac{1}{3} = \frac{7}{10};$$

$$\delta_{a.}(c, g, 8) = \frac{3}{5} \times \frac{1}{3} = \frac{1}{5};$$

$$\delta_{a.}(h, g, \{7, 8\}) = \frac{\sigma_{a(h,g,\{7,8\})} \cdot Flag(h, g, \{7, 8\})}{\sigma_{ag} \cdot |\{7, 8\}|} = \frac{2}{5} + \frac{1}{2} + \frac{\sigma_{a(h,g,\{7,8\})} \cdot |\{12, 13\}|}{\sigma_{a(g,f,\{12,13\})}} \cdot \delta_{a(g,f,\{12,13\})};$$

$$\delta_{a.}(h, g, 7) = \frac{1}{2} + \frac{2}{5} \times \frac{1}{2} = \frac{7}{10};$$

$$\delta_{a.}(h, g, 8) = \frac{2}{5} \times \frac{1}{2} = \frac{1}{5}.$$

The temporal edge dependency of other compressed edges and original temporal edges can be computed similarly. Note that for VFTP, the compressed outgoing edges e_c of the source vertex s should be handled specially. Only the largest timestamp is reserved, and the number of local VFTPs from s to e_c is set to 1 in the following process. For example, in Fig. 4(b), when traversing $(h, g, 7, 8)$ via $(a, h, \{4, 5\})$ which is the compressed outgoing edge of the source vertex a, $P_a(h, g, 7, 8) = (a, h, 5)$ rather than $P_a(h, g, 7, 8) = (a, h, \{4, 5\})$ and $\sigma_{a(a,h,5)} = 1$, because $(a, h, 4)$ is dominated by $(a, h, 5)$ by Definition 11.

v	b	c	h	g	f
$d(a,v)$	0	2	0	2	7
σ_{av}	2	2	2	2	5

$e_c=(u,v,\tau_v)$	$(a,b,1)$	$(a,b,5)$	$(a,h,\{4,5\})$	$(h,g,\{7,8\})$
$d(a,e_c)$	0	0	0	1
σ_{ae_c}	1	1	2	2
$Flag(e_c)$	1	1	1	1
$P_a(e_c)$	Ø	Ø	Ø	$(a,h,5)$

$e_c=(u,v,\tau_v)$	$(b,c,\{2,3\})$	$(b,c,6)$	$(c,g,\{7,8,10\})$	$(g,f,\{12,13\})$
$d(a,e_c)$	2	2	2	3
σ_{ae_c}	2	1	3	10
$Flag(e_c)$	1	1	1	1
$P_a(e_c)$	$(a,b,1)$	$(a,b,5)$	$(b,c,6)$	$(c,g,\{7,8,10\})$ $(h,g,\{7,8\})$

Fig. 5 Information maintained in forward scanning from a in Fig. 4(b)

Time and space complexities of optimized TEBC (abbr. OEBC). OEBC converts the input temporal graph $G=(V,E,T)$ of the TEBC algorithm into the compressed graph $G_c=(V_c,E_c,T_c)$ and performs edge dependency calculations using Lemma 3. While the whole processing of OEBC is still similar TEBC, which also requires forward and backward scanning. Hence the time complexity and space complexity are the same as those of TEBC except that $|V|$ and $|E|$ change to $|V_c|$ and $|E_c|$, respectively.

6 Experimental evaluation

6.1 Experiment settings

Datasets We employ 13 real datasets. Table 2 summarizes the statistics of the datasets used, where $|V|$, $|E|$, $|T|$, and $|E_c|$ are the number of vertices, temporal edges, different timestamps, and compressed edges, respectively. Hypertext, highschool-2012, sfhh-conference and highschool-2013 are from SocioPatterns. Highscool-2012 and highscool-2013 correspond to the contacts and friendship relations between students in a high school in Marseilles, France. Hypertext represents the dynamic network of face-to-face proximity of ~110 conference attendees at the ACM Hypertext 2009 conference. Sfhh-conference describes the face-to-face interactions of 405 participants at the 2009 SFHH conference in Nice, France. CollegeMsg, user-action, mathoverflow, sx-askubuntu, stackoverflow and sx-superuser are from SNAP. CollegeMsg is comprised of private messages sent on an online social network at the University of California, Irvine. User-action dataset represents the actions taken by users on a popular MOOC platform. Mathoverflow, sx-askubuntu, and sx-superuser are temporal networks of interactions on the stack exchange website Math Overflow, Ask Ubuntu, and superuser, respectively. Facebook-wosn, User-tag, and digg are from Konect. Facebook-wosn represents the friendships between a part of Facebook users. User-tag is taken from the official BibSonomy dump and represents tag assignment relationships. Digg represents votes on stories by users of Digg.

Methods We compare TECC with the state-of-the-art method TGC [19], and compare TEBC, Optimized TEBC (abbr. OEBC) with the state-of-the-art method TGB [36] which only supports VSTP. Note that TGC and TGB are used for vertex closeness and betweenness centrality computation, respectively, hence we converted them to ECC and EBC computation methods.

Table 2 Statistics of the datasets used

Datasets	$\|V\|$	$\|E\|$	$\|T\|$	$\|E_c\|$
hypertext	113	20,818	5,246	9,358
highschool-2012	180	45,047	11,273	12,632
collegeMsg	1,899	59,835	58,911	50,538
sfhh-conference	403	70,261	3,509	36,644
highschool-2013	327	188,508	7,375	64,312
facebook-wosn	55,387	335,708	333,923	335,708
user-action	7,047	411,749	345,600	202,937
mathoverflow	24,818	506,550	505,784	461,793
sx-askubuntu	159,316	964,437	960,866	916,342
stackoverflow	545,196	1,301,942	1,154	1,301,941
sx-superuser	194,085	1,443,340	1,437,199	1,379,206
user-tag	204,673	2,555,080	307,531	626,600
Digg	139,409	3,018,198	1,750,609	3,012,539

General setup All methods were implemented on the multi-thread framework OpenMP with C++, and run on a Ubuntu machine with 256 G memory and an Intel(R) Xeon(R) Silver 4314 2.40 GHz CPU. The optimizer option "O3" was used. The number of threads was set to 48 by default because 48 is sufficient to enable good performance, as verified in Section 6.2.

In the following, we evaluate the performance of TECC, TEBC, and OEBC. Note that we use bold values in the tables to highlight the best results in the same temporal path type, "–" to indicate that an algorithm cannot return the result within 12 hours, and "OOM" to represent that an algorithm runs out of memory. To clearly indicate the type of temporal paths on which the method is based, we add the path type suffix after the algorithm name, e.g., TECC-ESTP, TEBC-VSTP.

6.2 Performance of TECC and TEBC

Evaluating TECC. The result of TECC against TGC is illustrated in Table 3. The first observation is that, regardless of the path type, TECC performs much better than TGC on all datasets due to the effective strategies enabled by Definition 9 and Observations 1 and 2. To be specific, for ESTP, TECC is faster than TGC by 1.2 to 15 times; for EFTP, TECC outperforms TGC by 2 to 12 times; for EETP, TECC exceeds TGC by 2 to 17 times. TGC cannot return the results within 12 hours on massive datasets. The second observation is that, TECC (or TGC) based on ESTP is faster than that based on EETP and EFTP; and TECC (or TGC) based on EETP and EFTP are comparable. This is because, ESTP-based methods compute the minimum lengths from an edge to other vertices by a simple BFS paradigm. While EETP and EFTP-based methods require $O(\log|V|)$ time to insert (resp. remove) t-labels into (resp. from) the priority queue, as analyzed in Section 4.

Evaluating TEBC. Table 4 depicts the result of TEBC and OEBC against TGB based on VSTP. Table 5 depicts the result of TEBC and OEBC based on VFTP and VETP. Note that, we only report TGB-VSTP because TGB cannot be extended to support VETP and VFTP. The first observation is that, TEBC and OEBC dramatically outperform TGB. TGB runs out of memory on massive datasets. In particular, as shown in Table 4, on user-action, the computation costs of OEBC and TEBC are 566 and 397 times less than that of TGB. The reason is that, OEBC and TEBC employ the labeling-based forward-backward scanning strategy and temporal edge dependency formulas proposed by Lemmas 2 to 4 to iteratively accumulate EBC values, resulting in reduced computation cost. TGB takes up a lot of memory to build the predecessor graph. It easily leads to out-of-memory when computing massive temporal graphs. The second observation is that, no matter which type of temporal path, OEBC is more efficient than TEBC on all datasets because of the edge compression method and improved dependency formulas derived in Lemmas 3 and 4. In Particular, on highschool-2013, the number of compressed edges is 3 times less than that of

Table 3 TECC computation time (in second)

Datasets	TECC-ESTP	TGC-ESTP	TECC-EFTP	TGC-EFTP	TECC-EETP	TGC-EETP
hypertext	**0.12**	0.27	**0.24**	1.22	**0.22**	1.21
highschool-2012	**0.15**	0.92	**0.33**	1.49	**0.29**	1.57
collegeMsg	**2.7**	5.36	**5.15**	13.51	**5.15**	13.45
sfhh-conference	**0.96**	4.29	**1.54**	5.97	**1.53**	6.27
highschool-2013	**0.62**	4.86	**2.35**	14.8	**2.39**	14.25
facebook-wosn	**17.45**	68.56	**91.25**	363.52	**91.23**	364.01
user-action	**0.13**	2.22	**0.49**	6.26	**0.36**	6.39
mathoverflow	**164.91**	705.65	**1626.58**	7104.14	**1421.14**	6294.24
sx-askubuntu	**5924.07**	7205.75	**11670.54**	–	**10876.1**	–
stackoverflow	**1632.42**	4041.97	**8923.51**	–	**8394.12**	–
sx-superuser	**4378.41**	12663.12	**28149.3**	–	**26688.2**	–
user-tag	**7043.18**	17562.22	**15103.4**	–	**13821.1**	–
Digg	**3630.188**	13184.6	**8914.49**	–	**8297.71**	–

Table 4 TEBC computation time based on VSTP (in second)

Datasets	TEBC-VSTP	OEBC-VSTP	TGB-VSTP
hypertext	0.18	**0.05**	10.85
highschool-2012	0.4	**0.06**	38.53
collegeMsg	2.85	**2.55**	248.29
sfhh-conference	3.32	**1.35**	100.24
highschool-2013	2.56	**0.65**	280.37
facebook-wosn	103.41	**102.13**	OOM
user-action	4.52	**3.17**	1797.95
mathoverflow	1017.93	**896.98**	OOM
sx-askubuntu	9104.94	**7808.79**	OOM
stackoverflow	13321.4	**13318.3**	OOM
sx-superuser	28327.30	**24293.80**	OOM
user-tag	–	**15436.7**	OOM

Table 5 TEBC computation time based on VETP and VFTP (in second)

Datasets	TEBC-VFTP	OEBC-VFTP	TEBC-VETP	OEBC-VETP
hypertext	1.47	**0.35**	1.11	**0.33**
highschool-2012	3.76	**0.27**	2.51	**0.24**
collegeMsg	56.32	**39.28**	41.80	**31.75**
sfhh-conference	62.12	**18.81**	52.15	**15.51**
highschool-2013	55.08	**6.51**	40.77	**5.61**
facebook-wosn	1082.23	**1081.41**	819.86	**819.19**
user-action	82.86	**47.23**	63.25	**40.99**

original edges (as shown in Table 2), and OEBC is 4, 8, and 7 times faster than TEBC based on VSTP, VETP, and VFTP, respectively. However, on stackoverflow and facebook-wosn, OEBC and TEBC are comparable because only a few or no temporal edges are compressed. The last observation is that, similar to ECC computation, TEBC (or OEBC) based on VSTP is faster than that based on EETP and EFTP, and the computation cost of TEBC (or OEBC) based on EETP is close to that based on EFTP. This is because, seeking out VSTPs can be done in polynomial time. Nevertheless, finding all VETPs and VFTPs between any two vertices is a #p-hard problem [36], thus EBC computation based on VETP and VFTP has limited scalability and cannot run on massive datasets such as mathoverflow and sx-askubuntu. This is also consistent with the time complexity analysis in Section 5.

Effect of the number of threads This set of experiments investigates the impact of the number of threads. We vary the number of threads from 1 to 64, and illustrate the performance of TECC and OEBC based on different temporal path types(e.g., shortest temporal path (abbr. STP), fastest temporal path (abbr. FTP), earliest temporal path (abbr. ETP)) in Fig. 6. First, it is observed that, as the number of threads grows, memory usage increases linearly. This is because parallel computing requires more memory for data structure duplication. Additionally, we observe that the computation cost first significantly decreases and then gradually stabilizes or increases with the growth of the number of threads. This trend can be attributed to increased parallelism with more threads. However, beyond a certain thread number, the increased costs of thread switching and data consistency assurance equalize thread overhead and computational costs. As depicted in Fig. 6, setting the number of threads to 48 can achieve good performance.

6.3 Temporal edge ranking comparison

Table 6 shows Kendall's tau (KT) scores of TECC and TEBC based on three temporal path types. Kendall's tau correlation [37] is defined as (#concordant pairs - #discordant pairs) / #pairs, where #concordant pairs is the number of concordant pairs of temporal edge rankings; #discordant pairs is the number of discordant pairs of temporal edge rankings; and #pairs = $|E|(|E|-1)/2$ is the number of all pairs of temporal edges. KT scores range between −1 and +1, where 0 indicates that there is no correlation between the two rankings; 1 indicates that there is a completely positive correlation between the two rankings; and −1 indicates that there is a completely negative correlation between the two rankings. From columns 2 to 4 of Table 6, it is seen that, most KT scores are around 0 for TECC and TEBC in the same temporal path, which shows that TECC and TEBC nearly have no correlation. From columns 5 to 7, it is observed that for TECC, different path combinations exhibit a positive correlation to some extent on specific datasets. For example, on user-action, KT score of FTP and ETP is 0.34. From columns 8 to 10, we observe that KT scores are between −0.05 and 0.075, which shows that TEBC rankings based on different temporal path types are pairwise dissimilar.

6.4 Case study

Figure 7 shows the distribution of TECC and TEBC values based on STP on the part of hypertext, which is a temporal contact network with 16 vertices and 20 edges. In Fig. 7, the color shade and the thickness of the temporal edge are used to measure the importance of the edges. The thicker and darker

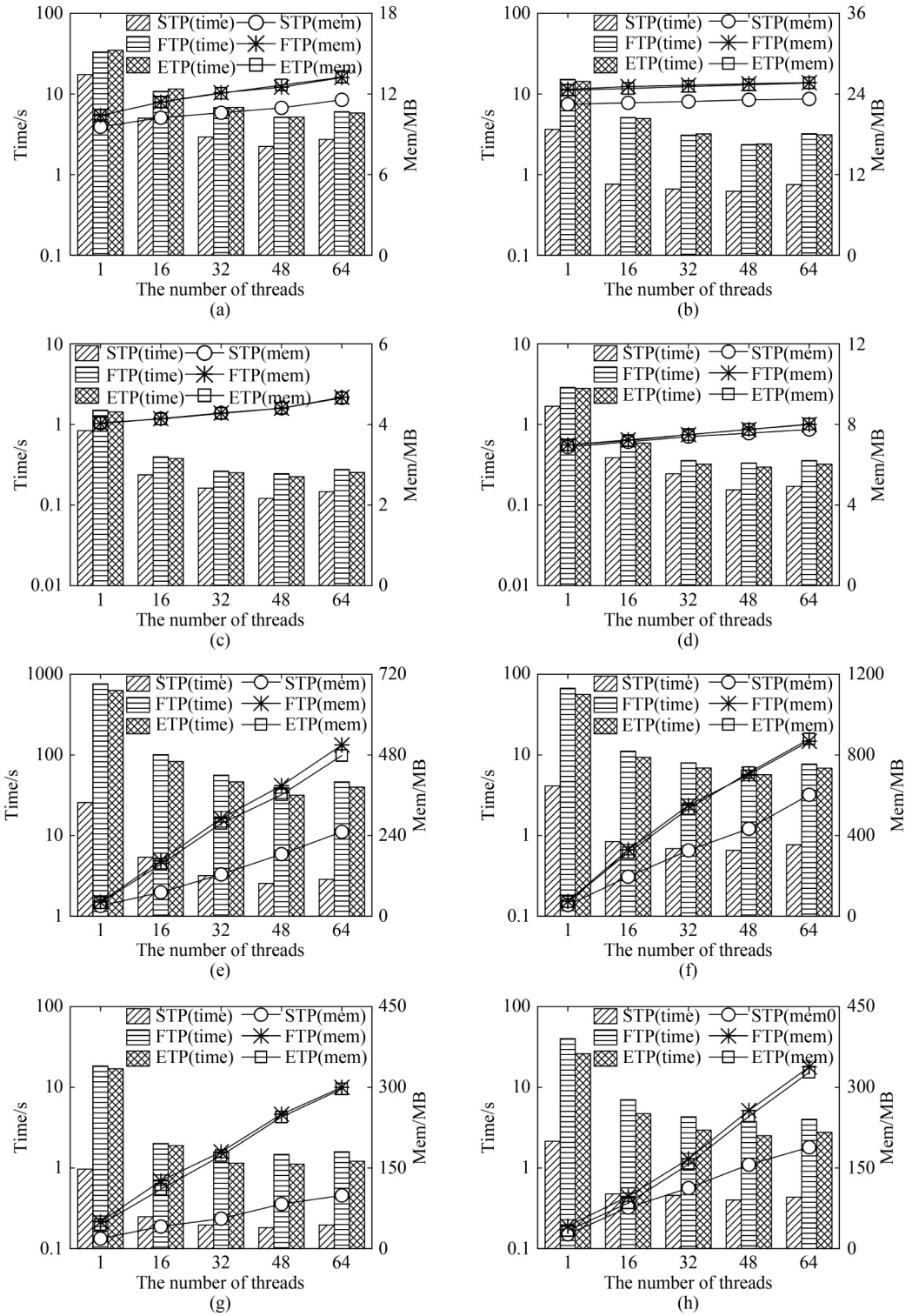


Fig. 6 TECC and TEBC performance costs vs. the number of threads. (a) TECC cost on collegeMsg; (b) TECC cost on highschool2013; (c) TECC cost on hypertext; (d) TECC cost on highschool2012; (e) TEBC cost on collegeMsg; (f) TEBC cost on highschool2013; (g) TEBC cost on hypertext; (h) TEBC cost on highschool2012

the edge is, the higher the TECC or TEBC value is, and the more critical the edge would be. In Fig. 7(a), it is seen that edges with timestamps 2 and 35 have the highest TECC values, which indicates edges that are further out to the side and have smaller timestamps are more likely to reach other vertices. In Fig. 7(b), it is observed that edges with timestamps 64 and 67 have high TEBC values, which shows that edges near the "center" are more likely to be passed by other vertices. Figure 7 reflects the difference between these two measures of centrality, in which high TECC values can be the source of information transmission and high TEBC values can be the bridges connecting different vertices.

Table 6 Kendall's tau correlation for all pairs of temporal edge rankings

Datasets	TECC vs TEBC			TECC			TEBC		
	STP	FTP	ETP	STP vs FTP	STP vs ETP	FTP vs ETP	STP vs FTP	STP vs ETP	FTP vs ETP
Hypertext	9.55×10^{-2}	5.94×10^{-2}	5.71×10^{-2}	8.3×10^{-2}	20.12×10^{-2}	8.66×10^{-2}	7.02×10^{-2}	3.44×10^{-2}	6.12×10^{-2}
Highschool-2012	2.8×10^{-2}	2.99×10^{-2}	6.17×10^{-2}	8.06×10^{-2}	23.3×10^{-2}	7.98×10^{-2}	2.23×10^{-2}	-1.35×10^{-2}	3.21×10^{-2}
CollegeMsg	-8.19×10^{-2}	-6.2×10^{-2}	23.18×10^{-2}	23.9×10^{-2}	5.27×10^{-2}	25.12×10^{-2}	0.45×10^{-2}	-0.86×10^{-2}	7.46×10^{-2}
Highschool-2013	0.13×10^{-2}	-0.02×10^{-2}	3.26×10^{-2}	7.01×10^{-2}	18.7×10^{-2}	7.04×10^{-2}	0.11×10^{-2}	0.42×10^{-2}	0.55×10^{-2}
User-action	4.95×10^{-2}	-3.95×10^{-2}	3.56×10^{-2}	3.94×10^{-2}	6.16×10^{-2}	34.32×10^{-2}	-0.31×10^{-2}	7.46×10^{-2}	-4.94×10^{-2}

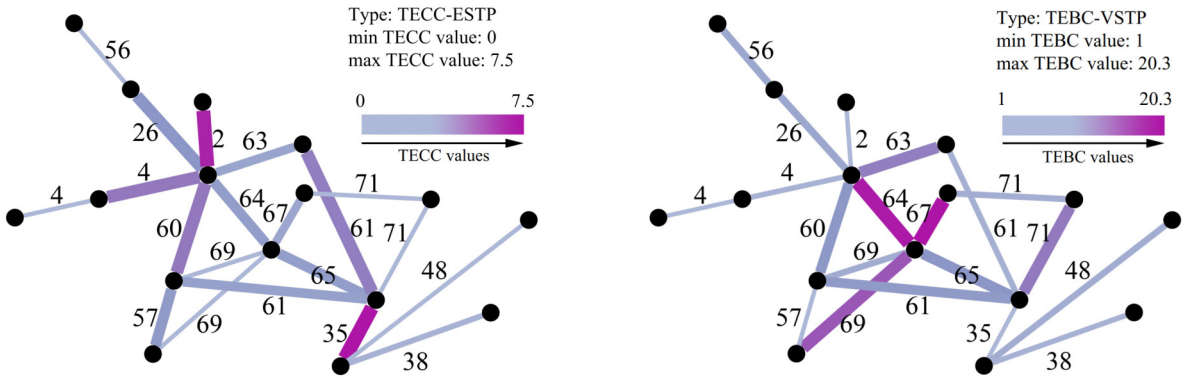


Fig. 7 Distribution of TECC and TEBC values on the part of hypertext. (a) Distribution of TECC values; (b) distribution of TEBC values

7 Conclusion

In this paper, we define label, label dominance relation, and propose multi-thread parallel labeling-based methods TECC and TEBC to efficiently compute ECC and EBC values of all edges w.r.t. three types of optimal temporal paths. For ECC computation, we provide a time segmentation strategy and two observations to aggregate some related temporal edges for uniform processing. For EBC computation, we derive temporal edge dependency formulas, a labeling-based forward-backward scanning strategy, and a compression-based optimization method that enable improved efficiency. An extensive experimental evaluation on 13 real data sets demonstrates that, compared with existing techniques, TECC and TEBC have higher time efficiency. In the future, it is of interest to investigate approximate ECC and EBC algorithms so as to further improve the performance. We also plan to develop efficient node or edge importance computation approaches for supporting graph updates, including vertex or edge insertions, deletions, and weight varying.

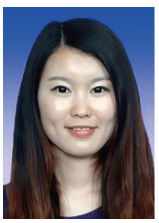
Acknowledgements This work was supported by the National Natural Science Foundation of China (Grant Nos. 62302451 and 62276233), the Natural Science Foundation of Zhejiang Province of China (No. LQ22F020018), and the Key Research Project of Zhejiang Province of China (No. 2023C01048).

Competing interests The authors declare that they have no competing interests or financial conflicts to disclose.

References

1. Freeman L C. Centrality in social networks conceptual clarification. Social Networks, 1978−1979, 1(3): 215–239
2. Girvan M, Newman M E. Community structure in social and biological networks. Proceedings of the National Academy of Sciences of the United States of America, 2002, 99(12): 7821–7826
3. Pournajar M, Zaiser M, Moretti P. Edge betweenness centrality as a failure predictor in network models of structurally disordered materials. Scientific Reports, 2022, 12(1): 11814
4. Simone A, Ridolfi L, Laucelli D, Berardi L, Giustolisi O. Centrality metrics for water distribution networks. EPiC Series in Engineering, 2018, 3: 1979–1988
5. Cuzzocrea A, Papadimitriou A, Katsaros D, Manolopoulos Y. Edge betweenness centrality: a novel algorithm for QoS-based topology control over wireless sensor networks. Journal of Network and Computer Applications, 2012, 35(4): 1210–1217
6. Ni P, Hanai M, Tan W J, Cai W. Efficient closeness centrality computation in time-evolving graphs. In: Proceedings of 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. 2019, 378−385
7. Tsalouchidou I, Baeza-Yates R, Bonchi F, Liao K, Sellis T. Temporal betweenness centrality in dynamic graphs. International Journal of Data Science and Analytics, 2020, 9(3): 257–272
8. Ghanem M. Temporal centralities: a study of the importance of nodes in dynamic graphs. Sorbonne Universités, Dissertation, 2018
9. Wu H, Cheng J, Huang S, Ke Y, Lu Y, Xu Y. Path problems in temporal graphs. Proceedings of the VLDB Endowment, 2014, 7(9): 721–732
10. Saxena A, Iyengar S. Centrality measures in complex networks: a survey. 2020, arXiv preprint arXiv: 2011.07190
11. Eppstein D, Wang J. Fast approximation of centrality. Journal of Graph Algorithms and Applications, 2004, 8(1): 39–45
12. Hoeffding W. Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association, 1963, 58(301): 13–30
13. Okamoto K, Chen W, Li X Y. Ranking of closeness centrality for large-scale social networks. In: Proceedings of the 2nd Annual International Workshop on Frontiers in Algorithmics. 2008, 186−195
14. Cohen E, Delling D, Pajor T, Werneck R F. Computing classic closeness centrality, at scale. In: Proceedings of the 2nd ACM Conference on Online Social Networks. 2014, 37−50

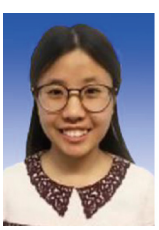
15. Olsen P W, Labouseur A G, Hwang J H. Efficient top-k closeness centrality search. In: Proceedings of the 30th IEEE International Conference on Data Engineering. 2014, 196−207
16. Guimarães A, Vieira A B, Silva A P C, Ziviani A. Fast centrality-driven diffusion in dynamic networks. In: Proceedings of the 22nd International Conference on World Wide Web. 2013, 821−828
17. Shao Z, Guo N, Gu Y, Wang Z, Li F, Yu G. Efficient closeness centrality computation for dynamic graphs. In: Proceedings of the 25th International Conference on Database Systems for Advanced Applications. 2020, 534−550
18. Oettershagen L, Mutzel P. Efficient top-k temporal closeness calculation in temporal networks. In: Proceedings of 2020 IEEE International Conference on Data Mining. 2020, 402−411
19. Oettershagen L, Mutzel P. Computing top-k temporal closeness in temporal networks. Knowledge and Information Systems, 2022, 64(2): 507–535
20. Bergamini E, Borassi M, Crescenzi P, Marino A, Meyerhenke H. Computing top-*k* closeness centrality faster in unweighted graphs. ACM Transactions on Knowledge Discovery from Data, 2019, 13(5): 53
21. Brandes U. A faster algorithm for betweenness centrality. The Journal of Mathematical Sociology, 2001, 25(2): 163–177
22. Erdős D, Ishakian V, Bestavros A, Terzi E. A divide-and-conquer algorithm for betweenness centrality. In: Proceedings of 2015 SIAM International Conference on Data Mining. 2015, 433−441
23. Sariyüce A E, Kaya K, Saule E, Çatalyürek Ü V. Graph manipulations for fast centrality computation. ACM Transactions on Knowledge Discovery from Data, 2017, 11(3): 26
24. Baglioni M, Geraci F, Pellegrini M, Lastres E. Fast exact computation of betweenness centrality in social networks. In: Proceedings of 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. 2012, 450−456
25. Kanwar K, Kaushal S, Kumar H, Gupta G, Khari M. BCDCN: a new edge centrality measure to identify and rank critical edges pertaining to SIR diffusion in complex networks. Social Network Analysis and Mining, 2022, 12(1): 49
26. De Meo P, Ferrara E, Fiumara G, Ricciardello A. A novel measure of edge centrality in social networks. Knowledge-Based Systems, 2012, 30: 136–150
27. Brandes U, Pich C. Centrality estimation in large networks. International Journal of Bifurcation and Chaos, 2007, 17(7): 2303–2318
28. Riondato M, Kornaropoulos E M. Fast approximation of betweenness centrality through sampling. In: Proceedings of the 7th ACM International Conference on Web Search and Data Mining. 2014, 413−422
29. Cousins C, Wohlgemuth C, Riondato M. Bavarian: betweenness centrality approximation with variance-aware rademacher averages. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. 2021, 196−206
30. Lee M J, Lee J, Park J Y, Choi R H, Chung C W. QUBE: a quick algorithm for updating betweenness centrality. In: Proceedings of the 21st International Conference on World Wide Web. 2012, 351−360
31. Green O, McColl R, Bader D A. A fast algorithm for streaming betweenness centrality. In: Proceedings of 2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing. 2012, 11−20
32. Kourtellis N, De Francisci Morales G, Bonchi F. Scalable online betweenness centrality in evolving graphs. IEEE Transactions on Knowledge and Data Engineering, 2015, 27(9): 2494–2506
33. Kas M, Wachs M, Carley K M, Carley L R. Incremental algorithm for updating betweenness centrality in dynamically growing networks. In: Proceedings of 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. 2013, 33−40
34. Bergamini E, Meyerhenke H, Staudt C L. Approximating betweenness centrality in large evolving networks. In: Proceedings of Meeting on Algorithm Engineering & Expermiments. 2015, 133−146
35. Hayashi T, Akiba T, Yoshida Y. Fully dynamic betweenness centrality maintenance on massive networks. Proceedings of the VLDB Endowment, 2015, 9(2): 48–59
36. Buß S, Molter H, Niedermeier R, Rymar M. Algorithmic aspects of temporal betweenness. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2020, 2084−2092
37. Knight W R. A computer method for calculating Kendall's tau with ungrouped data. Journal of the American Statistical Association, 1966, 61(314): 436–439



Tianming Zhang received the BS and MS degrees in computer science from Northeastern University, China in 2012 and 2014, respectively, and the PhD degree in computer science from Zhejiang University, China in 2020. She is currently a lecturer in the College of Computer Science, Zhejiang University of Technology, China. Her research interest includes graph data management and analysis.

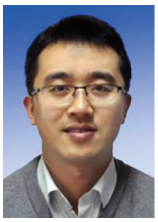
Jie Zhao is currently working toward the MS degree in the College of Computer Science, Zhejiang University of Technology, China. His main research interests include graph querying and processing.

Cibo Yu is currently working toward the BS degree in the College of Computer Science, Zhejiang University of Technology, China. His main research interests include graph analysis and mining.



Lu Chen received the PhD degree in computer science from Zhejiang University, China in 2016. She was an associate professor in Aalborg University, Denmark. She is currently a ZJU Plan 100 professor in the College of Computer Science, Zhejiang University, China. Her research interests include indexing and querying metric spaces, graph databases, and database usability.

Yunjun Gao (Member, IEEE) received the PhD degree in computer science from Zhejiang University, China in 2008. He is currently a professor in the College of Computer Science, Zhejiang University, China. His research interests include spatial and spatio-temporal databases, metric and incomplete/uncertain data management, graph databases, spatio-textual data processing, and database usability. He is a member of the ACM and the IEEE.



Bin Cao (Member, IEEE) received his PhD degree in computer science from Zhejiang University, China in 2013. He then worked as a research associate in Hongkong University of Science and Technology and Noah's Ark Lab, Huawei, China. He joined Zhejiang University of Technology, China in 2014, and is now an associate professor in the College of Computer Science. His research interests include spatio-temporal database and data mining.

Jing Fan received the BS, MS, and PhD degree in computer science from Zhejiang University, China in 1990, 1993, and 2003, respectively. She is currently a professor in the College of Computer Science, Zhejiang University of Technology, China. She is Vice-Director of Key Laboratory of Visual Media Intelligent Processing Technology of Zhejiang Province, China. Her current research interests include service computing, software middleware, virtual reality and visualization. She is a Director of China Computer Federation (CCF), and Member of CCF Technical Committee on Service Computing.

Ge Yu (Member, IEEE) received the PhD degree in computer science from the Kyushu University of Japan in 1996. He is currently a professor and the PhD supervisor at the Northeastern University of China. His research interests include distributed and parallel database, OLAP and data warehousing, data integration, graph data management, etc. He is a member of the IEEE Computer Society, ACM, and a fellow of the China Computer Federation (CCF).