Efficient Processing of Growing Temporal Graphs

Huanhuan Wu, Yunjian Zhao, James Cheng, Da Yan

Department of Computer Science and Engineering, The Chinese University of Hong Kong
{hhwu,yjzhao,jcheng,yanda}@cse.cuhk.edu.hk

Abstract. Temporal graphs are useful in modeling real-world networks. For example, in a phone call network, people may communicate with each other in multiple time periods, which can be modeled as multiple temporal edges. However, the size of real-world temporal graphs keeps increasing rapidly (e.g., considering the number of phone calls recorded each day), which makes it difficult to efficiently store and analyze the complete temporal graphs. We propose a new model, called *equal-weight damped time window model*, to efficiently manage temporal graphs. In this model, each time window is assigned a unified weight. This model is flexible as it allows users to control the tradeoff between the required storage space and the information loss. It also supports efficient maintenance of the windows as new data come in. We then discuss applications that use the model for analyzing temporal graphs. Our experiments demonstrated that we can handle massive temporal graphs efficiently with limited space.

1 Introduction

A temporal graph is a graph in which the relationship between vertices is not just modeled by an edge between them, but the time period when the relationship happens is also recorded. For example, two persons A and B talked on the phone in time periods $[t_1, t_2]$ and $[t_3, t_4]$ are modeled as two temporal edges, $(A, B, [t_1, t_2])$ and $(A, B, [t_3, t_4])$. An example of a temporal graph is shown in Figure 1(a).

Graphs are used ubiquitously to model relationships between objects in real world. However, the graph data in many applications are actually better to be modeled as temporal graphs. For example, in communication networks, including online social networks, messaging networks, phone call networks, etc., people communicate with each other in different time periods. Temporal graphs collected from these applications carry rich time information, and have been shown to possess many important time-related patterns that cannot be found from non-temporal graphs [8,9,10,13,17,19,25].

However, existing work overlooks one serious problem presented by temporal graphs in real world applications, that is, the number of temporal edges (or temporal records) can be extremely huge so that it becomes overly expensive to store and process a temporal graph. For example, in a temporal graph that models phone-call records, a person may talk on the phone many times in different time periods in a day, where each phone call is represented by a temporal edge with the corresponding time period. The total number of temporal edges accumulated over time for all persons can easily become overwhelming. Note that while the number of temporal edges usually increases at a steady rate over time, the number of vertices, on the other hand, does not increase too much over time after passing the growth stage.

The problem in the above example is actually a real problem presented to us by a telecommunications operator, who collects phone-call and messaging records represented as a temporal graph that becomes too large over time for them to manage (millions to tens of millions of new temporal edges added each day). While analyzing only a short recent window of the data is useful, the telecom operator is also very keen in storing and analyzing the temporal graph over a long period of time (e.g., in recent years), and possibly the entire history, in an efficient way. Motivated by this, we propose a new model to efficiently manage a temporal graph.

Our new model considers the input temporal graph as a continuous stream, which captures how the temporal graph is collected in real-life applications (e.g., new call/message records are accumulated in the order of the calling/sending time). However, the sheer size of the stream over the entire time history renders analysis (and even storage) of the original temporal graph too costly. To address this problem, we consider a *damped time window model* (also called *tilted time window*) [5], where a decay function is applied to depreciate the importance of records in an older window. However, the windows defined by existing damped time window models do not have a unified weight and hence the importance of records in different windows cannot be easily compared. For example, which of the following patterns is more important: a pattern that A and B communicated 10 times in a recent window (e.g., last week), or a pattern that A and B communicated 10,000 times in an older window (e.g., last year)?

We design a new damped time window model that gives a unified weight to each time window, called *equal-weight damped time window*, and represents the temporal graph falling into each window (i.e., a time period) as a weighted graph. The weighted graphs from different time windows can then be compared and analyzed.

The main contributions of our work are as follows:

- Our equal-weight damped time window distributes a unified weight to each time window, which makes it easy to compare different time windows.
- Our model can handle massive temporal graphs with limited space requirement, and support efficient graph analysis with little information loss.
- The equal-weight design in our model also leads to natural and efficient update maintenance of the entire window (within a bounded storage space).
- We present an application that analyzes the connectivity of a temporal graph with our new model. More applications such as community finding can be found in [23]. We verified the effectiveness and efficiency of our method by extensive experiments on large temporal graph datasets.

Outline. Section 2 presents the equal-weight time window model. Section 3 discusses one application based on our model. Section 4 reports experimental results. Section 5 discusses related work. Section 6 concludes the paper.

2 Equal-Weight Damped Time Window

Different window models have been proposed for processing a data stream. Among which, the *landmark window model* [14] considers the entire history of a stream without distinguishing the importance of recent and old records, while the *sliding window*

model [6] focuses on the most recent window only. Our work is motivated by application needs from a telecom operator that requires to analyze historical data while giving more importance to recent data. For this purpose, the *damped time window model* [5] seems to suit the requirement. We introduce our damped time window model in this section, and discuss its difference with existing ones.

We first define the notations related to a temporal graph. Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a temporal graph, where $\mathbb{V}$ is the set of vertices and $\mathbb{E}$ is the set of edges in $\mathbb{G}$. An edge $e \in \mathbb{E}$ is a quadruple $(u, v, [t_i, t_j])$, where $u, v \in \mathbb{V}$, and $[t_i, t_j]$ is the time that e is *active*. We focus our discussion on undirected temporal graphs, while we note that it is not difficult to extend our method to directed temporal graphs.

2.1 The Weight Function

In a damped time window model, a decaying weight function is used to depreciate the importance of a record over time. In the setting of a temporal graph, we use such a function to assign weight to temporal edges in the graph. We first present the weight density function as follows.

Definition 1 (Weight density function). *Let t_τ be the current time. The weight density of a record at time t (with respect to t_τ) is defined as*

$$f(t) = e^{\lambda(t-t_\tau)},$$

where $\lambda \geq 0$ is a decaying constant.

Note that $t \leq t_\tau$, and t is a time in the past if $t < t_\tau$.

In Definition 1, $f(t)$ is an exponentially decaying function, which is used throughout the paper as it has been widely adopted [11,24]. But $f(t)$ can also be defined differently (e.g., as a linear decaying function) depending on the application. Based on $f(t)$, we define our weight function as follows.

Definition 2 (Weight function). *The weight of a temporal edge $(u, v, [t_1, t_2])$ is given as the integral*

$$F(t_1, t_2) = \int_{t_1}^{t_2} f(t)dt.$$

Let $W = [t_x, t_y]$ be a given time window. With the weight function, we represent the part of a temporal graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ that falls into W as a weighted graph G_W defined as follows.

Definition 3 (Weighted graph). *The weighted graph of a temporal graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ within a time window $W = [t_x, t_y]$ is given by $G_W = (V_W, E_W, \Pi_W)$, where:*

- $V_W = \mathbb{V}$,
- $E_W = \{(u, v) : (u, v, [t_i, t_j]) \in \mathbb{E}_W\}$, *where* $\mathbb{E}_W = \{(u, v, [t_i, t_j]) \in \mathbb{E} : [t_i, t_j] \cap [t_x, t_y] \neq \emptyset\}$,
- Π_W *is a function that assigns each edge* $e = (u, v) \in E_W$ *a weight* $\Pi_W(e) = \sum_{(u,v,[t_i,t_j]) \in \mathbb{E}_W} F(\max(t_i, t_x), \min(t_j, t_y))$.

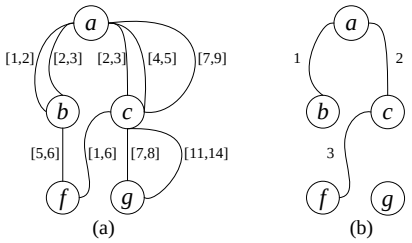

Fig. 1. (a) a **temporal graph** $\mathbb{G}$, and **(b)** the **weighted graph** $G_{[2,5]}$

Example 1. Figure 1(a) shows a temporal graph and Figure 1(b) shows the corresponding weighted graph $G_{[2,5]}$ within the time window $[2, 5]$. For simplicity, we assume that $\lambda = 0$ and hence $f(t) = 1$. Thus, the weight of edge (a, b) is $F(2, 3) = 1$, the weight of edge (a, c) is $F(2, 3) + F(4, 5) = 2$, and the weight of edge (c, f) is $F(2, 5) = 3$. The weight of all other edges is 0.

2.2 The Equal-Weight Window Model

Next, we determine the size of each window in a data stream given the weight function.

Existing damped time window model [5] usually sets the sizes of the windows in a stream by an exponentially increasing function (e.g., 2^0T, 2^1T, 2^2T, 2^3T, ..., where the windows are disjoint and the most recent window has a size 2^0T), or by the lengths of conventional time units (e.g., hour, day, month, year, ...). These window size settings may seem to be intuitive, but they are primarily designed for mining frequent itemsets from a stream and are not suitable for our problem of handling a temporal graph stream (see more discussion in "Advantages of the new model" at the end of this subsection). We introduce an equal-weight scheme as follows.

Let $[t_0, t_\tau]$ be the time period of the entire stream up to the current time t_τ. To limit the space requirement for handling a large temporal graph, we divide the stream into θ windows for a given constant number θ. We first define the equal-weight window condition as follows.

Definition 4 (Equal-weight window condition).
Consider that the probability distribution of any edge being active at any time follows a uniform distribution. Under this distribution, the equal-weight condition is satisfied if the stream is divided into θ windows such that the weighted graph of each window is the same in expectation.

Intuitively, Definition 4 states that *if the probability of any edge being active does not change over time, then the weight of the edge should not change in any of the θ windows*.

Let the time periods of the θ windows be $[t_0, t_1]$, $[t_1, t_2]$, ..., $[t_{\theta-1}, t_\tau]$. Then, applying Definition 4, we have

$$\int_{t_0}^{t_1} f(t)dt = \int_{t_1}^{t_2} f(t)dt = ... = \int_{t_{\theta-1}}^{t_\tau} f(t)dt = \frac{1}{\theta}\int_{t_0}^{t_\tau} f(t)dt.$$

Take $f(t) = e^{\lambda(t-t_\tau)}$. We first determine t_1 as follows:

$$\int_{t_0}^{t_1} f(t)dt = \frac{1}{\theta}\int_{t_0}^{t_\tau} f(t)dt$$

$$\Rightarrow \frac{1}{\lambda}(e^{\lambda(t_1-t_\tau)} - e^{\lambda(t_0-t_\tau)}) = \frac{1}{\theta\lambda}(e^{\lambda(t_\tau-t_\tau)} - e^{\lambda(t_0-t_\tau)})$$

$$\Rightarrow \theta(e^{\lambda t_1} - e^{\lambda t_0}) = e^{\lambda t_\tau} - e^{\lambda t_0}$$

$$\Rightarrow e^{\lambda t_1} = \frac{e^{\lambda t_\tau} + (\theta-1)e^{\lambda t_0}}{\theta}$$

$$\Rightarrow t_1 = \frac{1}{\lambda}\ln\frac{e^{\lambda t_\tau} + (\theta-1)e^{\lambda t_0}}{\theta}.$$

Similarly, we obtain t_i, where $1 \leq i \leq \theta - 1$, as follows:

$$t_i = \frac{1}{\lambda}\ln\frac{i \times e^{\lambda t_\tau} + (\theta-i)e^{\lambda t_0}}{\theta}.$$

Based on the above analysis, we define *equal-weight damped time window model* as follows.

Definition 5 (Equal-weight damped window model). *Given a stream that spans the time period* $[t_0, t_\tau]$*, and an integer* θ*,* equal-weight damped time window model *divides the stream into* θ *windows spanning time periods* $[t_0, t_1]$, $[t_1, t_2]$, ..., $[t_{\theta-1}, t_\tau]$*, where* $t_i = \frac{1}{\lambda}\ln\frac{i\times e^{\lambda t_\tau}+(\theta-i)e^{\lambda t_0}}{\theta}$*, for* $1 \leq i \leq \theta - 1$.

Based on Definition 5, we obtain θ weighted graphs derived from the temporal graph that falls into each of the θ windows in the stream. The value of θ is determined by users, which controls the space requirement and the efficiency of graph analysis, as well as the degree of information loss (from the original temporal graph to the θ weighted graphs). The larger the value of θ, the finer is the granularity of the windows and the less is the information loss, but also the more is the memory space needed.

Advantages of the new model. The equal-weight damped time window model has the following advantages: (A1) it is a generalization of existing damped time window models; (A2) it gives equal importance to each window, which makes it easy to compare the graphs from different windows; and (A3) it provides a systematical way for update maintenance of the windows.

For (A1), by defining an appropriate weight density function, we can apply our proposed equal-weight scheme to compute the size of each window for existing damped time window models. Take the logarithmic tilted-time window model as an example, where an exponentially increasing function (e.g., 2^0T, 2^1T, 2^2T, 2^3T, ...) is used. Assume that the time span of the entire stream is $[0, 2^\theta - 1]$, then the weight density function is defined as follows:

$$f(t) = \begin{cases} 1, & 0 \leq t < 2^{\theta-1} \\ 2, & 2^{\theta-1} \leq t < 2^{\theta-1} + 2^{\theta-2} \\ ..., & ... \\ 2^{\theta-1}, & 2^\theta - 2 \leq t \leq 2^\theta - 1 \end{cases}$$

For (A2), if the weights of an edge (u, v) in two different windows W_1 and W_2 are the same, then it implies that the probability of (u, v) being active remains the same in W_1 and W_2. Now if the probability of (u, v) being active is higher in W_1, then apparently the weight of (u, v) in W_1 is also higher than that in W_2. This may not be true if existing damped time windows are used unless we define an appropriate $f(t)$ function for them, and apply our scheme proposed in this section to determine the window sizes.

For (A3), we show that our model provides a systematical way for update maintenance of the windows in Section 2.3.

2.3 Window Maintenance

As time goes on, new temporal edges are created and the windows need to be updated. We devise an update scheme for our window model as follows.

Let $[t_0, t_1]$, $[t_1, t_2]$, ..., $[t_{\theta-1}, t_\theta]$ denote the θ existing windows, and $[t_\theta, t_{\theta+1}]$ denote the new window. As the current time changes from $t_\tau = t_\theta$ to $t'_\tau = t_{\theta+1}$, the weight density function $f(t)$ changes from $f(t) = e^{\lambda(t-t_\theta)}$ to $f(t) = e^{\lambda(t-t_{\theta+1})}$. Lemmas 1 and 2 state the change needed.

Due to the space limitation, the proofs for all the lemmas and theorems are given in the full version of this paper [23].

Lemma 1. *If the current time changes from t_τ to t'_τ, for any temporal edge whose weight w is last updated at time t_τ, the weight should be updated as follows:*

$$w \leftarrow w \times e^{\lambda(t_\tau - t'_\tau)}.$$

Lemma 2. *Given a weighted graph $G = (V, E, \Pi)$ of any window, if the current time changes from t_τ to t'_τ, the weight w of each edge in E which is computed at time t_τ should be updated as follows*

$$w \leftarrow w \times e^{\lambda(t_\tau - t'_\tau)}.$$

Lemma 2 shows that it is simple to update the weighted graphs of the existing windows as new windows are created in the stream. However, we still need to determine at what point a new window, i.e., $[t_\theta, t_{\theta+1}]$, should be created in the stream, which is to determine $t_{\theta+1}$. Following our discussion in Section 2.2, we have

$$\int_{t_{\theta-1}}^{t_\theta} f(t)dt = \int_{t_\theta}^{t_{\theta+1}} f(t)dt$$

$$\Rightarrow e^{\lambda t_\theta} - e^{\lambda t_{\theta-1}} = e^{\lambda t_{\theta+1}} - e^{\lambda t_\theta}$$

$$\Rightarrow t_{\theta+1} = \frac{1}{\lambda} \ln(2e^{\lambda t_\theta} - e^{\lambda t_{\theta-1}}).$$

Similarly, we can also compute the windows that are to follow in the stream, i.e., $[t_{\theta+1}, t_{\theta+2}]$, ... However, in this way, the number of windows keeps increasing, and the size of a new window (i.e., the time span of the window) becomes smaller and smaller.

To solve these issues, we propose to merge windows to bound the number of windows in the stream within the range $[\theta, 2\theta]$. Specifically, when the number of windows reaches 2θ, we merge every two consecutive windows into one window. In this way, every window in the stream after merging still satisfies the equal-weight window condition. In fact, we can also merge more than two windows into a single window if necessary.

3 Window-Based Network Analysis

We now discuss network analysis based on the equal-weight damped time window model, which we illustrate using an application of connectivity analysis in this section. We also discuss other applications (e.g., community finding) and give a list of open problems about analyzing large temporal graphs using our model, but due to the space limitation we present the details in the full version of this paper [23].

Let G_1, G_2, ..., G_θ denote the θ weighted graphs derived from the θ windows in the stream.

3.1 Connectivity Analysis

Given a weighted graph $G = (V, E, \Pi)$ of a window in the stream (defined in Definition 3, and here the window W is omitted for simplicity), we define a measure of connectivity between two vertices u and v in G as follows.

Definition 6 (Connectivity). *Let $\mathbb{P}(u, v) = \{P(u, v) : P(u, v)$ is a path from u to v in $G\}$. The* connectivity *of a path $P(u, v)$, denoted by $\gamma(P(u, v))$, is defined as the minimum edge weight among the edges on $P(u, v)$. The* connectivity *between u and v, denoted by $\gamma(u, v)$, is defined as $\gamma(u, v) = \max\{\gamma(P(u, v)) : P(u, v) \in \mathbb{P}(u, v)\}$.*

Since the weight of each edge in a weighted graph indicates the strength of relationship (or interaction, communication, etc.) between the two end points in the corresponding temporal graph within the time span of the window, the value of $\gamma(u, v)$ reflects the connectivity between u and v within the window, for example, the amount of information that can be passed between u and v via any path within the time span.

Given a *connectivity query* $\gamma(u, v)$, we can answer it using an algorithm similar to Dijkstra's algorithm, as shown in Algorithm 1. Algorithm 1 uses a maximum priority queue Q to keep the current largest connectivity value, $c[x]$, of a path from u to a visited vertex $x \in V$. The algorithm starts from one of the query vertices, u, greedily grows the paths by extending to u's neighbors, and then further grows to the neighbors' neighbors until reaching the other query vertex v. During the greedy process, the $c[x]$ value of a vertex x is updated whenever a larger connectivity value from u to x is found. At each iteration, the vertex with maximum $c[.]$ is extracted from Q to update the $c[.]$ values of its neighbors.

We now show the correctness and complexity (the proof is given in [23]).

Theorem 1. *Algorithm 1 correctly computes the connectivity value $\gamma(u, v)$ in $O((|E| + |V|) \log |V|)$ time.*

Algorithm 1: Compute $\gamma(u, v)$

Input : A weighted graph $G = (V, E, \Pi)$, two query vertices u and v
Output: $\gamma(u, v)$

```
1  Initialize c[u] ← ∞, c[x] ← 0 for every vertex x ∈ V \ {u};
2  Let Q be a maximum priority queue, where an element of Q is a pair (x, c[x]) and c[x]
   being the key;
3  Initialize Q by inserting a single element (u, c[u]);
4  while Q is not empty do
5      (x, c[x]) ← Extract-Max(Q);
6      if x = v then
7          Goto Line 12;
8      foreach neighbor vertex, y, of x do
9          if c[y] < min(c[x], Π(x, y)) then
10             c[y] ← min(c[x], Π(x, y));
11             If y is not in Q, push (y, c[y]) into Q; otherwise, update c[y] in Q;
12 return γ(u, v) = c[v];
```

The complexity of Algorithm 1, even if Fibonacci heap is used, is too high to process a connectivity query online. One may pre-compute the connectivity values for all pairs of vertices. However, the space complexity of this method is $O(|V|^2)$, and the pre-computation requires $O((|E| + |V|)|V| \log |V|)$ time, both of which are impractical for handling a large graph. We propose a more efficient way to process connectivity queries, with linear index space.

First, we compute a *maximum spanning tree*, *MaxST*, of the weighted graph G. Without loss of generality, we assume G is connected. If not, we can consider each connected component of G separately. A MaxST has the cut property. A *cut* is a partition of the vertex set of a graph into two disjoint subsets. We say that *an edge crosses the cut* if it has one endpoint in each subset of the partition. The *cut property* states that for any cut C in the graph, if the weight of an edge e crossing C is larger than the weights of all the other edges crossing C, then e must be contained in every MaxST.

Given a MaxST, T, there is a unique path connecting any two vertices in T. Let $\gamma_T(u, v)$ denote the connectivity value between u and v in the MaxST T. Based on the cut property, we have the following lemma (the proof is given in [23]).

Lemma 3. *Given a MaxST T of a weighted graph G, $\gamma(u, v) = \gamma_T(u, v)$, for any pair of vertices u and v in G.*

Based on Lemma 3, a connectivity query $\gamma(u, v)$ can be answered by first finding the unique path between u and v in the MaxST T, and then returning the minimum edge weight on the path. The query time complexity is $O(|V|)$, which is much better than that of Algorithm 1. Next, we show that we can further reduce the querying time complexity to $O(1)$ time.

We first introduce the concept of *Cartesian tree* [18], which is a binary tree derived from a sequence of numbers. Given an array A of n numbers ($A[0]$ to $A[n-1]$), the

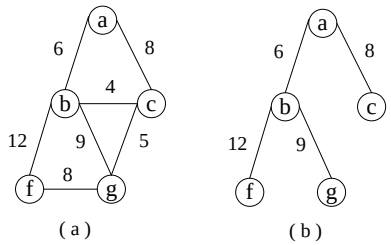

Fig. 2. (a) a **weighted graph** G, and **(b)** a **MaxST** T

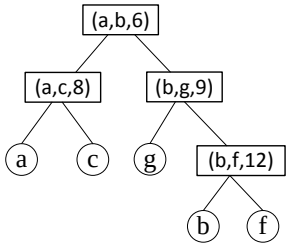

Fig. 3. The **Cartesian tree** C_T of T in Figure 2(b)

root of the Cartesian tree is the minimum number among all the n numbers. Let $A[i]$ be the minimum number, i.e., the root. Then, its left subtree is computed recursively on the numbers $A[0]$ to $A[i-1]$, while its right subtree is computed recursively on the numbers $A[i+1]$ to $A[n-1]$.

We construct a Cartesian tree, C_T, based on a MaxST T. The root node of C_T is the edge with the minimum weight among all the edges in T. Then, by removing this edge, T will be partitioned into two subtrees. Following a similar procedure, we can recursively construct the left and right subtrees of the root node. When removing an edge (u, v), if u (and/or v) is not an end point of any remaining edges in T, then we also create a leaf node u (and/or v) as the child of the node (u, v) in C_T. Thus, the set of leaf nodes in the tree C_T corresponds to the set of vertices in T.

Based on the Cartesian tree C_T, given a connectivity query $\gamma(u, v)$, we first find the lowest common ancestor (LCA) of the two leaves u and v in C_T, and then return the weight of the edge in T that corresponds to the LCA. The following example demonstrates the concepts of MaxST, Cartesian tree, and how to answer a connectivity query.

Example 2. Figure 2(a) shows a weighted graph G and Figure 2(b) shows a MaxST T of G. It is easy to verify $\gamma(u, v) = \gamma_T(u, v)$. For example, $\gamma(c, f) = 6$ in G and $\gamma_T(c, f) = 6$ in T. Figure 3 shows the Cartesian tree C_T of T. The root node of C_T is the edge $(a, b, 6)$, since this edge is the one with the minimum weight in T. Removing $(a, b, 6)$ partitions T into two components $\{a, c\}$ and $\{b, f, g\}$. Following a similar procedure recursively, we obtain C_T. The leaves of C_T are the vertices in T. Then, to find the connectivity value between any two vertices, we find the LCA of these two vertices in C_T. For example, given a connectivity query $\gamma(f, g)$, we find that the

edge $(b, g, 9)$ is the LCA of the leaves f and g in C_T. Thus, we return 9 as the answer for $\gamma(f, g)$. It is easy to verify that the answer is correct.

Now, we give the complexity of processing a connectivity query and of constructing the index (the proof is given in [23]).

Theorem 2. *A connectivity query $\gamma(u, v)$ can be answered in $O(1)$ time with an index using $O(|V|)$ space, and the index construction time is $O((|E| + |V|) \log |V|)$.*

Given the θ weighted graphs G_1, G_2, ..., G_θ from the θ windows in the stream, we define the connectivity between u and v in the entire θ windows as $\Gamma(u, v) = \min\{\gamma_1(u, v), \ldots, \gamma_\theta(u, v)\}$, where $\gamma_i(u, v)$ is the connectivity value $\gamma(u, v)$ in the weighted graph G_i, for $1 \leq i \leq \theta$. Since θ is a constant, the query $\Gamma(u, v)$ can be answered in constant time with indexes of size linear to the number of vertices.

3.2 Queries on a Random Window

Besides the need of analysis on a temporal graph in the whole time window, users may also be interested in analyzing the graph in any time period. For example, user A is interested in time window $[1, 20]$, while user B is interested in time range $[10, 40]$. To satisfy each user's need, the naive way is to store the complete temporal graph and extract the temporal subgraph from the required time range, which is not practical due to the massive size of the complete graph. We discuss how to efficiently obtain a weighted graph of any time period based on the equal-weight damped time window model.

Given a random window $W = [t_x, t_y]$, we are required to return $G_W = (V, E_W, \Pi_W)$. Given θ weighted graphs, $G_1 = (V, E_1, \Pi_1)$, $G_2 = (V, E_2, \Pi_2)$, ..., $G_\theta = (V, E_\theta, \Pi_\theta)$, we return an approximate weighted graph G'_W of G_W as follows.

Let $t_i < t_x \leq t_{i+1}$ and $t_j \leq t_y < t_{j+1}$. First, we return an approximate weighted graph $G'_{[t_x, t_{i+1}]}$ of $G_{[t_x, t_{i+1}]}$. $G'_{[t_x, t_{i+1}]} = (V, E'_{[t_x, t_{i+1}]}, \Pi'_{[t_x, t_{i+1}]})$ is computed as follows:

- $E'_{[t_x, t_{i+1}]} = E_{i+1}$,
- $\Pi'_{[t_x, t_{i+1}]}(e) = \Pi_{i+1}(e) \times \frac{\int_{t_x}^{t_{i+1}} f(t)dt}{\int_{t_i}^{t_{i+1}} f(t)dt}$, for each $e \in E'_{[t_x, t_{i+1}]}$.

In other words, $G'_{[t_x, t_{i+1}]}$ is computed based on $G_{i+1} = (V, E_{i+1}, \Pi_{i+1})$ in expectation. Similarly, we compute an approximate weighted graph $G'_{[t_j, t_y]}$ of $G_{[t_j, t_y]}$. Then, we have $G'_W = (V, E'_W, \Pi'_W)$ as follows:

- $E'_W = E'_{[t_x, t_{i+1}]} \cup E_{i+2} \cup \ldots \cup E'_{[t_j, t_y]}$,
- $\Pi'_W(e) = \Pi'_{[t_x, t_{i+1}]}(e) + \Pi_{i+2}(e) + \ldots + \Pi'_{[t_j, t_y]}(e)$, for each $e \in E'_W$.

4 Experimental Results

We evaluated the usefulness of our equal-weight window model by showing the quality of the θ weighted graphs obtained based on the model, and the efficiency and quality

Table 1. Real temporal graphs (K = 10^3)

Dataset	$\|\mathbb{V}\|$	$\|\mathbb{E}\|$	$d_{avg}(v, \mathbb{G})$	$\|T_{\mathbb{G}}\|$
phone	1,237	338,008,540	273,248.62	3,369
arxiv	28,094	9,193,606	327.24	2,337
elec	8,298	214,028	25.79	101,063
enron	87,274	2,282,904	26.16	220,364
facebook	46,953	1,730,624	36.86	867,939
lastfm	174,078	38,254,660	219.76	17,498,009
email	168	164,613	979.84	57,842
conflict	118,101	5,903,522	49.99	312,457

of graph analysis based on these weighted graphs. We also verified the efficiency of dynamic update maintenance and the scalability of our method, where the results of them are reported in [23] due to the space limitation. All the experiments were run on a Linux machine with an Intel 3.3GHz CPU and 16GB RAM. All the programs were implemented in C++ and complied using G++ 4.8.2.

We used 8 real temporal graphs for our experiments, as shown in Table 1, where we list the number of vertices and edges in each graph $\mathbb{G}$, the average degree in $\mathbb{G}$ (denoted by $d_{avg}(v, \mathbb{G})$), and the number of distinct time instances in $\mathbb{G}$ (denoted by $|T_{\mathbb{G}}|$). The phone graph consists of call records in Ivory Coast [1], where the call records were collected over a span of 150 days. The other 7 graphs were obtained from the Koblenz Large Network Collection (http://konect.uni-koblenz.de/), where one large temporal graph was selected from each of the following 7 categories: arxiv-HepPh (arxiv) from the arxiv networks; elec from the network of English Wikipedia; enron from the email networks; facebook-links (facebook) from the facebook network; lastfm-band (lastfm) from the music website last.fm; radoslaw-email (email) from the internal email communication network between employees of a mid-sized manufacturing company; wikiconflict (conflict) indicating conflicts between users of Wikipedia.

4.1 Results on Weighted Graph Construction

In this experiment, we evaluated the space requirement and the construction time of the θ weighted graphs for each of the temporal graphs, and then we measured the quality of the weighted graphs. We tested θ from 10 to 50. We set the value of $\lambda = 10^{-x}$ for the weight density function given in Definition 1, where $10^x \leq |T_{\mathbb{G}}| < 10^{x+1}$, that is, $\lambda = 10^{-\lfloor \log_{10} |T_{\mathbb{G}}| \rfloor}$. For example, for the phone graph, $\lambda = 10^{-3}$.

Space requirement. We first report the space requirement for the θ weighted graphs, as a percentage of the original temporal graph shown in Figure 4. As the value of θ increases, the total size of the θ weighted graphs also increases. But the rate of increase is slow. For graphs with high average degree, the total size of the θ weighted graphs is only a small percentage of the original temporal graph. For example, for the phone graph, even the total size of 50 weighted graphs is less than 10% of the original temporal graph. We emphasize that for temporal graphs, the set of vertices remains relatively stable while the number of temporal edges grows linearly over time, and thus the result

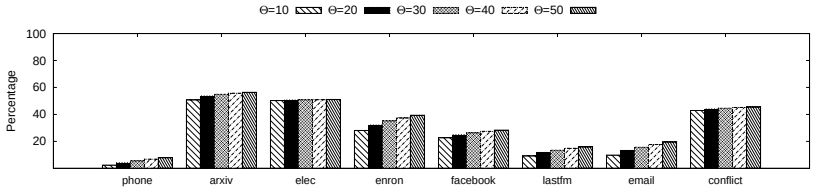

Fig. 4. The total size of the θ weighted graphs compared with the original temporal graph $\mathbb{G}$

Table 2. Construction time of θ weighted graphs (in seconds)

Dataset	$\theta = 10$	$\theta = 20$	$\theta = 30$	$\theta = 40$	$\theta = 50$
phone	130.3067	137.6559	143.6432	148.6535	153.1762
arxiv	4.0591	4.2070	4.3718	4.4772	4.5788
elec	0.1110	0.1168	0.1229	0.1266	0.1292
enron	0.8419	0.9031	0.9600	1.0041	1.0473
facebook	0.6245	0.6743	0.6996	0.7325	0.7581
lastfm	12.6525	13.2842	14.0147	14.6400	15.5061
email	0.0511	0.0548	0.0575	0.0607	0.0617
conflict	2.8762	2.9693	3.0447	3.1341	3.2189

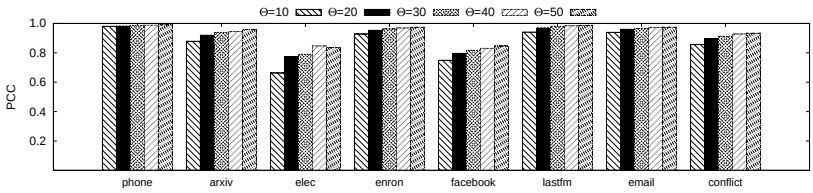

Fig. 5. PCC between G'_W and G_W for different θ

verifies that our method can handle large temporal graphs as they grow over time, with small space requirement.

Construction time. Table 2 reports the time taken to read the temporal graphs from disk and construct the corresponding θ weighted graphs, for different values of θ. The construction is fast for all graphs as we only need to scan the graphs once, regardless of the value of θ. The construction time increases as θ increases because more weighted graphs need to be constructed, but the rate of increase is slow as scanning the original temporal graph dominates the cost.

Quality of results. Next, we examine the quality of the weighted graphs. To do this, we constructed a weighted graph, G_W, directly from the original temporal graph within a time window W, as defined in Definition 3. We also constructed an approximate weighted graph G'_W of G_W from the θ weighted graphs as discussed in Section 3.2. Then, we compared G_W and G'_W.

We computed G_W and G'_W for 100 randomly generated windows, $W = [t_x, t_y]$, where we ensured that W is a valid window by ensuring $t_x < t_y$. We use *Pearson correlation coefficient* (*PCC*) to measure the degree of linear correlation between G'_W and G_W, and report the results in Figure 5.

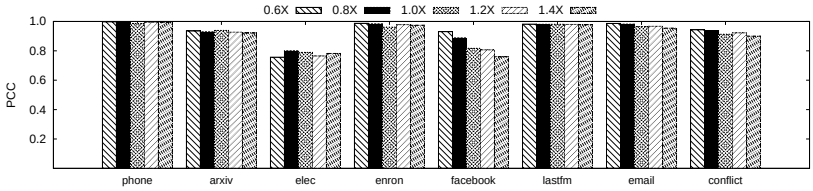

Fig. 6. PCC between G'_W and G_W for different λ ($\theta = 30$)

Table 3. Average query processing time of connectivity queries (in milliseconds)

	$\theta = 10$		$\theta = 20$		$\theta = 30$		$\theta = 40$		$\theta = 50$	
	Index	Online	Index	Online	Index	Online	Index	Online	Index	Online
phone	0.0041	24.0973	0.0059	42.0291	0.0082	56.7214	0.0095	71.6784	0.0116	85.7897
arxiv	0.0045	21.6051	0.0081	19.8686	0.0127	18.9631	0.0160	18.3361	0.0188	17.8703
elec	0.0029	0.5434	0.0057	0.5460	0.0083	0.6127	0.0108	0.6866	0.0140	0.7909
enron	0.0049	5.6191	0.0103	6.2276	0.0132	6.8081	0.0186	7.7741	0.0230	8.7257
facebook	0.0052	5.7762	0.0095	5.5650	0.0141	5.6486	0.0185	5.9857	0.0231	6.4586
lastfm	0.0062	31.5103	0.0122	34.3018	0.0201	40.0901	0.0249	43.6900	0.0331	46.0317
email	0.0004	0.1239	0.0008	0.1856	0.0016	0.2295	0.0022	0.2649	0.0030	0.3012
conflict	0.0052	15.5628	0.0098	14.3276	0.0149	13.6799	0.0195	14.3760	0.0255	15.2124

The result shows that we obtain high PCC values in most of the cases, which implies that analysis conducted on the approximate graph G'_W shares similar patterns/trends with that conducted on the exact graph G_W (we further verify this point by applying the application in Section 3. The results can be found in [23] (e.g., Figure 8 in [23]) which lead to a similar conclusion as Figure 5. As θ increases from 10 to 50, the PCC values also increase, verifying that a larger θ leads to less information loss and hence higher correlation between G'_W and G_W. For a number of graphs, the PCC values are close to 1. The results are particularly impressive for the phone graph, for which the space requirement is also very small as shown in Figure 4.

Next, we tested the effect of different values of λ. In all the other experiments, we set $\lambda = 10^{-\lfloor \log_{10} |T_{\mathbb{G}}| \rfloor}$ as default. In this experiment, we tested λ at 0.6, 0.8, 1.0, 1.2, and 1.4 of its default value, and fixed $\theta = 30$. The result, as reported in Figure 6, shows that the PCC values are not much affected by the change in λ, and in all cases the PCC values are high.

Efficiency of graph analysis. We also evaluated the efficiency of using the θ weighted graphs for connectivity analysis. We varied θ from 10 to 50, and tested 1000 randomly generated connectivity queries. We used the index presented in Section 3.1 to answer the queries, and compared with the online algorithm given in Algorithm 1. We denote these two methods by *Index* and *Online*, respectively. Table 3 reports the average processing time per query. The result shows that *Index* is more than 3 orders of magnitude faster than *Online*, verifying the efficiency of our method. The index construction time and the index size are also small, which are linear to the number of vertices (as shown Table 1).

Due to the space limitation, we report more results in [23], which show that our method is efficient and effective for core community analysis in temporal graphs, is fast in dynamic update maintenance, and has good scalability.

5 Related Work

Much of the work on temporal graphs, also called time-varying graphs or timetable graphs, was related to temporal paths. Temporal paths have been applied to study the connectivity of a temporal graph [9], the information latency in a temporal network [10], small-world behavior [17], and to find temporal connected components [16]. Temporal paths have also been used to define metrics for temporal network analysis, such as temporal efficiency and temporal clustering coefficient [15,16], and temporal closeness [13]. Algorithms for computing temporal paths were discussed in [19,20,25]. Indexing method for answering reachability and time-based path queries in a temporal graph was proposed in [22]. Diversified subgraph pattern mining in a temporal graph was introduced in [30]. Core decomposition in a large temporal graph was studied in [21]. Readers can also refer to more comprehensive surveys on temporal graphs [4,8,12], and more related work can be found in the full version of this paper [23].

There are also works on storing temporal graphs in a compact way [2,3,7]. In [2], a compressed suffix array strategy was proposed to store temporal graphs. In [3], two data structures, compact adjacency sequence and compact events ordered by time, were proposed to represent temporal graphs. However, all these methods need to store each temporal edge, and their performance is not better than the *gzip* compression.

6 Conclusions

We proposed the *equal-weight damped time window* model for processing massive growing temporal graphs. Our model allows users to set the number of windows to trade off between the required space and the information loss. Based on this model, we presented an application of connectivity analysis to analyze the temporal graph. We conducted comprehensive experiments to verify the usefulness and efficiency of our method for analyzing large temporal graphs. As for future work, we plan to explore how to integrate the proposed time window model into our prior work on distributed graph processing systems [26,27,28,29] to analyze massive dynamic temporal graphs.

Acknowledgements. We thank the reviewers for their valuable comments. The authors are supported by the Hong Kong GRF 2150851 and 2150895, ITF 6904079, MSRA grant 6904224, and CUHK Grants 3132964 and 3132821.

References

1. V. D. Blondel, M. Esch, C. Chan, F. Clérot, P. Deville, E. Huens, F. Morlot, Z. Smoreda, and C. Ziemlicki. Data for development: the D4D challenge on mobile phone data. *CoRR*, abs/1210.0137, 2012.
2. N. R. Brisaboa, D. Caro, A. Fariña, and M. A. Rodríguez. A compressed suffix-array strategy for temporal-graph indexing. In *SPIRE*, pages 77–88, 2014.
3. D. Caro, M. A. Rodríguez, and N. R. Brisaboa. Data structures for temporal graphs based on compact sequence representations. *Inf. Syst.*, 51:1–26, 2015.
4. A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

5. Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-dimensional regression analysis of time-series data streams. In *VLDB*, pages 323–334, 2002.
6. M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.
7. G. de Bernardo, N. R. Brisaboa, D. Caro, and M. A. Rodríguez. Compact data structures for temporal graphs. In *DCC*, page 477, 2013.
8. P. Holme and J. Saramäki. Temporal networks. *CoRR*, abs/1108.1780, 2011.
9. D. Kempe, J. M. Kleinberg, and A. Kumar. Connectivity and inference problems for temporal networks. *J. Comput. Syst. Sci.*, 64(4):820–842, 2002.
10. G. Kossinets, J. M. Kleinberg, and D. J. Watts. The structure of information pathways in a social communication network. In *KDD*, pages 435–443, 2008.
11. J. Lai, C. Wang, and P. S. Yu. Dynamic community detection in weighted graph streams. In *SDM*, pages 151–161, 2013.
12. M. Müller-Hannemann, F. Schulz, D. Wagner, and C. D. Zaroliagis. Timetable information: Models and algorithms. In *ATMOS*, pages 67–90, 2004.
13. R. K. Pan and J. Saramäki. Path lengths, correlations, and centrality in temporal networks. *Phys. Rev. E*, 84:016105, 2011.
14. C. Perng, H. Wang, S. R. Zhang, and D. S. P. Jr. Landmarks: a new model for similarity-based pattern querying in time series databases. In *ICDE*, pages 33–42, 2000.
15. J. Tang, M. Musolesi, C. Mascolo, and V. Latora. Temporal distance metrics for social network analysis. In *WOSN*, pages 31–36, 2009.
16. J. Tang, M. Musolesi, C. Mascolo, and V. Latora. Characterising temporal distance and reachability in mobile and online social networks. *Computer Communication Review*, 40(1):118–124, 2010.
17. J. Tang, S. Scellato, M. Musolesi, C. Mascolo, and V. Latora. Small-world behavior in time-varying graphs. *Physical Review E*, 81(5):055101, 2010.
18. J. Vuillemin. A unifying look at data structures. *Commun. ACM*, 23(4):229–239, 1980.
19. H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu. Path problems in temporal graphs. *PVLDB*, 7(9):721–732, 2014.
20. H. Wu, J. Cheng, Y. Ke, S. Huang, Y. Huang, and H. Wu. Efficient algorithms for temporal path computation. *IEEE Trans. Knowl. Data Eng.*, 28(11):2927–2942, 2016.
21. H. Wu, J. Cheng, Y. Lu, Y. Ke, Y. Huang, D. Yan, and H. Wu. Core decomposition in large temporal graphs. In *IEEE International Conference on Big Data*, pages 649–658, 2015.
22. H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke. Reachability and time-based path queries in temporal graphs. In *ICDE*, pages 145–156, 2016.
23. H. Wu, Y. Zhao, J. Cheng, and D. Yan. Efficient processing of growing temporal graphs. 2016. `http://www.cse.cuhk.edu.hk/%7ejcheng/papers/tm_tr.pdf`.
24. W. Xie, Y. Tian, Y. Sismanis, A. Balmin, and P. J. Haas. Dynamic interaction graphs with probabilistic edge decay. In *ICDE*, pages 1143–1154, 2015.
25. B.-M. B. Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. Found. Comput. Sci.*, 14(2):267–285, 2003.
26. D. Yan, J. Cheng, Y. Lu, and W. Ng. Blogel: A block-centric framework for distributed computation on real-world graphs. *PVLDB*, 7(14):1981–1992, 2014.
27. D. Yan, J. Cheng, Y. Lu, and W. Ng. Effective techniques for message reduction and load balancing in distributed graph computation. In *WWW*, pages 1307–1317, 2015.
28. D. Yan, J. Cheng, M. T. Özsu, F. Yang, Y. Lu, J. C. S. Lui, Q. Zhang, and W. Ng. A general-purpose query-centric framework for querying big graphs. *PVLDB*, 9(7):564–575, 2016.
29. F. Yang, J. Li, and J. Cheng. Husky: Towards a more efficient and expressive distributed computing framework. *PVLDB*, 9(5):420–431, 2016.
30. Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, and J. C. S. Lui. Diversified temporal subgraph pattern mining. In *SIGKDD*, pages 1965–1974, 2016.