Inferring Tie Strength in Temporal Networks*

Lutz Oettershagen[1](✉), Athanasios L. Konstantinidis[2], Giuseppe F. Italiano[2]

[1] Institute of Computer Science, University of Bonn, Bonn, Germany
lutz.oettershagen@cs.uni-bonn.de
[2] Luiss University, Rome, Italy
{akonstantinidis, gitaliano}@luiss.it

Abstract. Inferring tie strengths in social networks is an essential task in social network analysis. Common approaches classify the ties as *weak* and *strong* ties based on the *strong triadic closure (STC)*. The STC states that if for three nodes, A, B, and C, there are strong ties between A and B, as well as A and C, there has to be a (weak or strong) tie between B and C. So far, most works discuss the STC in static networks. However, modern large-scale social networks are usually highly dynamic, providing user contacts and communications as streams of edge updates. *Temporal networks* capture these dynamics. To apply the STC to temporal networks, we first generalize the STC and introduce a weighted version such that empirical a priori knowledge given in the form of edge weights is respected by the STC. The weighted STC is hard to compute, and our main contribution is an efficient 2-approximative streaming algorithm for the weighted STC in temporal networks. As a technical contribution, we introduce a fully dynamic 2-approximation for the minimum weight vertex cover problem, which is a crucial component of our streaming algorithm. Our evaluation shows that the weighted STC leads to solutions that capture the a priori knowledge given by the edge weights better than the non-weighted STC. Moreover, we show that our streaming algorithm efficiently approximates the weighted STC in large-scale social networks.

Keywords: Triadic Closure, Temporal Network, Tie Strength Inference.

1 Introduction

Due to the explosive growth of online social networks and electronic communication, the automated inference of tie strengths is critical for many applications, e.g., advertisement, information dissemination, or understanding of complex human behavior [11,20]. Users of large-scale social networks are commonly connected to hundreds or even thousands of other participants [26,30]. It is the typical case that these ties are not equally important. For example, in a social network, we can be connected with close friends as well as casual contacts. Since a pioneering work of Granovetter [12], the topic of tie strength inference has

* Giuseppe F. Italiano is partially supported by MUR, the Italian Ministry for University and Research, under PRIN Project AHeAD (Efficient Algorithms for HArnessing Networked Data).

gained increasing attention fueled by the advent of online social networks and ubiquitous contact data. Nowadays, ties strength inference in social networks is an extensively studied topic in the graph-mining community [10,20,38]. A recent work by Sintos and Tsaparas [39] introduced the *strong triadic closure (STC)* property, where edges are classified as either *strong* or *weak*—for three persons with two *strong* ties, there has to be a *weak* or *strong* third tie. Hence, if person A is strongly connected to B, and B is strongly connected to C, A and C are at least weakly connected. The intuition is that if A and B are good friends, and B and C are good friends, A and C should at least know each other.

(a) Example for optimal weighted STC. The edge weights correspond the amount of communications.

(b) Example for optimal non-weighted STC. Ignoring the weights leads to three strong edges.

Fig. 1: Example for the difference between weighted and non-weighted STC. Strong edges are highlighted in red, and weak edges are dashed.

We first generalize the ideas of [39] such that edge weights representing empirical tie strength are included in the computation of the STC. The idea is to consider edge weights that correspond to the empirical strength of the tie, e.g., the frequency or duration of communication between two persons. If this weight is high, we expect the tie to be strong and weak otherwise. However, we still want to fulfill the STC, and simple thresholding would not lead to correct results. Figure 1 shows an example where we have a small social network consisting of four persons A, B, C, and D. In Figure 1a, the edge weights correspond to some empirical a priori information of the tie strength like contact frequency or duration, e.g., A and B chatted for ten hours and B and D for only one hour. The optimal weighted solution classifies the edges between A and B as well as between C and D as strong (highlighted in red). However, if we ignore the weights, as shown in Figure 1b, the optimal (non-weighted) solution has three strong edges. Even though the non-weighted solution has more strong edges, the weighted version agrees more with our intuition and the empirical a priori knowledge.

We employ this generalization of the STC for inferring the strength of ties between nodes in temporal networks. A temporal network consists of a fixed set of vertices and a chronologically ordered stream of appearing and disappearing temporal edges, i.e., each temporal edge is only available at a specific discrete point in time. Temporal networks can naturally be used as models for real-life scenarios, e.g., communication [4,7], contact [33], and social networks [14,17,31]. In contrast to static graphs, temporal networks are not simple in the sense that

between each pair of nodes, there can be several temporal edges, each corresponding to, e.g., a contact or communication at a specific time [18]. Hence, there is no one-to-one mapping between edges and ties. Given a temporal network, we map it to a weighted static graph such that the edge weights are a function of the empirical tie strength. We then classify the edges using the weighted STC, respecting the a priori information given by the edge weights.

A major challenge is that the weighted STC is hard to compute and real-world temporal networks are often provided as large or possibly infinite streams of graph updates. To tackle this computational challenge, we employ a sliding time window approach and introduce a streaming algorithm that can efficiently update a 2-approximate of the minimum weighted STC, i.e., the problem that asks for the minimum total weight of weak edges. Our contributions are:

1. We generalize the STC for weighted graphs and apply the weighted STC for determining tie strength in temporal networks. To this end, we use temporal information to infer the edge strengths of the underlying static graph.
2. We provide a streaming algorithm to efficiently approximate the weighted STC over time with an approximation factor of two. As a technical contribution, we propose an efficient dynamic 2-approximation for the minimum weighted vertex cover problem, a key ingredient of our streaming algorithm.
3. Our evaluation shows that the weighted STC leads to strong edges with higher weights consistent with the given empirical edge weights. Furthermore, we show the efficiency of our streaming algorithm, which is orders of magnitude faster than the baseline.

Omitted proofs and the appendix can be found in the extended version [32].

2 Related Work

There are various studies on predicting the strength of ties given different features of a network, e.g., [10,44]. However, these works do not classify edges with respect to the STC. In contrast, our work is based on the STC property, which was introduced by Granovetter [12]. An extensive analysis of the STC can be found in the book of Easley and Kleinberg [6]. Recently, Sintos and Tsaparas [39] proposed an optimization problem by characterizing the edges of the network as strong or weak using only the structure of the network. They proved that the problem of maximizing the strong edges is NP-hard, although they provided two approximation algorithms to solve the dual problem of minimizing the weak edges. In the following works, the authors of [13,24,25] focused on restricted networks to further explore the complexity of STC maximization.

Rozenshtein et al. [38] discuss the STC with additional community connectivity constraints. Adriaens et al. [1] proposed integer linear programming formulations and corresponding relaxations. Very recently, Matakos et al. [29] proposed a new problem that uses the strong ties of the network to add new edges and increase its connectivity. The mentioned works only consider static networks

and do not include edges weights in the computation of the STC. We propose a weighted variant and use it to infer ties strength in temporal networks.

Even though temporal networks are a quite recent research field, there are some comprehensive surveys introduce the notation, terminology, and applications [18,27]. Additionally, there are systematic studies into the complexity of well-known graph problems on temporal networks (e.g. [16,21]). The problem of finding communities and clusters, which can be considered as a related problem, has been studied on temporal networks [5,41]. Furthermore, Zhou et al. [45] studied dynamic network embedding based on a model of the triadic closure process, i.e., how open triads evolve into closed triads. Huang et al. [19] studied the formation of closed triads in dynamic networks. The authors of [2] introduce a probabilistic model for dynamic graphs based on the triadic closure.

Wei et al. [42] introduced a dynamic $(2+\varepsilon)$-approximation for the minimum weight vertex cover problem with $\mathcal{O}(\log n/\varepsilon^2)$ amortized update time based on a vertex partitioning scheme [3]. However, the algorithm does not support updates of the vertex weights, which is an essential operation in our streaming algorithm.

3 Preliminaries

An *undirected* (static) *graph* $G = (V, E)$ consists of a finite set of nodes V and a finite set $E \subseteq \{\{u, v\} \subseteq V \mid u \neq v\}$ of undirected edges. We use $V(G)$ and $E(G)$ to denote the sets of nodes and edges, respectively, of G. The set $\delta(v) = \{e = \{v, w\} \mid e \in E(G)\}$ contains all edges incident to $v \in V(G)$, and we use $d(v) = |\delta(v)|$ to denote the degree of $v \in V$. An *edge-weighted* undirected graph $G = (V, E, w_E)$ is an undirected graph with additional weight function $w : E \to \mathbb{R}$. Analogously, we define a *vertex-weighted* undirected graph $G = (V, E, w_V)$ with a weight function for the vertices $w : V \to \mathbb{R}$. If the context is clear, we omit the subscript of the weight function.

A *wedge* is defined as a triplet of nodes $u, v, w \in V$ such that $\{\{u, v\}, \{v, w\}\} \subseteq E$ and $\{u, w\} \notin E$. We denote such a wedge by $(v, \{u, w\})$, and with $\mathcal{W}(G)$ the set of wedges in a graph G. Next, we define the *weighted wedge graph*. The non-weighted version is also known as the *Gallai* graph [8].

Definition 1. *Let $G = (V, E, w_E)$ be a edge-weighted graph. The* weighted wedge graph $W(G) = (U, H, w_V)$ *consists of the vertex set $U = \{n_{uv} \mid \{u, v\} \in E\}$, the edges set $H = \{\{n_{uv}, n_{vw}\} \mid (v, \{u, w\}) \in \mathcal{W}(G)\}$, and the vertex weight function $w_V(n_{uv}) = w_E(\{u, v\})$.*

Temporal Networks A *temporal network* $\mathcal{G} = (V, \mathcal{E})$ consists of a finite set of nodes V, a possibly infinite set $\mathcal{E}$ of undirected *temporal edges* $e = (\{u, v\}, t)$ with u and v in V, $u \neq v$, and *availability time* (or *timestamp*) $t \in \mathbb{N}$. For ease of notation, we may denote temporal edges $(\{u, v\}, t)$ with (u, v, t). We use $t(e)$ to denote the availability time of e. We do not include a duration in the definition of temporal edges, but our approaches can easily be adapted for temporal edges with duration parameters. We define the underlying static, weighted, *aggregated* graph $A_\phi(\mathcal{G}) = (V, E, w)$ of a temporal network $\mathcal{G} = (V, \mathcal{E})$ with the edges set

$E = \{\{u, v\} \mid (\{u, v\}, t) \in \mathcal{E}\}$ and edge weight function $w : E \to \mathbb{R}$. The edge weights are given by the function $\phi : 2^{\mathcal{E}} \to \mathbb{R}$ such that $w(e) = \phi(\mathcal{F}_e)$ with $\mathcal{F}_e = \{e \mid (e, t) \in \mathcal{E}\}$. We discuss various weighting functions in Section 5.1. Finally, we denote the lifetime of a temporal network $\mathcal{G} = (V, \mathcal{E})$ with $T(\mathcal{G}) = [t_{min}, t_{max}]$ with $t_{min} = \min\{t \mid e = (u, v, t) \in \mathcal{E}\}$ and $t_{max} = \max\{t \mid e = (u, v, t) \in \mathcal{E}\}$.

Strong Triadic Closure Given a (static) graph $G = (V, E)$, we can assign one of the labels *weak* or *strong* to each edge in $e \in E$. We call such a labeling a *strong-weak labeling*, and we specify the labeling by a subset $S \subseteq E$. Each edge $e \in S$ is called *strong*, and $e \in E \setminus S$ *weak*. The *strong triadic closure (STC)* of a graph G is a strong-weak labeling $S \subseteq E$ such that for any two strong edges $\{u, v\} \in S$ and $\{v, w\} \in S$, there is a (weak or strong) edge $\{u, w\} \in E$. We say that such a labeling *fulfills* the strong triadic closure. In other words, in a strong triadic closure there is no pair of strong edges $\{u, v\}$ and $\{v, w\}$ such that $\{u, w\} \notin E$. Consequently, a labeling $S \subseteq E$ fulfills the STC if and only if at most one edge of any wedge in $\mathcal{W}(G)$ is in S, i.e., there is no wedge with two strong edges [39]. The decision problem for the STC is denoted by MaxSTC and is stated as follows: Given a graph $G = (V, E)$ and a non-negative integer k. Does there exist $S \subseteq E$ that fulfills the strong triadic closure and $|S| \geq k$?

See Figure 1 in Appendix C.1 for examples of a temporal, aggregated, and wedge graph, and the STC.

4 Weighted Strong Triadic Closure

Let $G = (V, E, w)$ be a graph with edge weights reflecting the importance of the edges. We determine a *weighted* strong triadic closure that takes the weights of the edges by the importance given by w into account. To this end, let $S \subseteq E$ be a strong-weak labeling. The labeling S fulfills the weighted STC if (1) for any two strong edges $\{u, v\}, \{v, w\} \in S$ there is a (weak or strong) edge $\{u, w\} \in E$, i.e., fulfills the unweighted STC, and (2) maximizes $\sum_{e \in S} w(e)$.

The corresponding decision problem WeightedMaxSTC has as input a graph $G = (V, E)$ and $U \in \mathbb{R}$, and asks for the existence of a strong-weak labeling that fulfills the strong triadic closure and for which $\sum_{e \in S} w(e) \geq U$. Sintos and Tsaparas [39] showed that MaxSTC is NP-complete using a reduction from Maximum Clique. The reduction implies that we cannot approximate the MaxSTC with a factor better than $\mathcal{O}(n^{1-\epsilon})$. Because MaxSTC is a special case of WeightedMaxSTC, these negative results also hold for the latter.

Instead of maximizing the weight of strong edges, we can equivalently minimize the weight of weak edges resulting in the corresponding problem WeightedMinSTC[3]. Here, we search a strong-weak labeling that fulfills the STC and minimizes the weight of the edges not in S. Both the maximation and the minimization problems can be solved exactly using integer linear programming (ILP). We provide the corresponding ILP formulations in Appendix A. The advantage of WeightedMinSTC is that we can obtain a 2-approximation.

[3] We use WeightedMinSTC for the decision and the optimization problem in the following if the context is clear.

To approximate WeightedMinSTC in an edge-weighted graph $G = (V, E, w)$, we first construct the weighted wedge graph $W(G) = (V_W, E_W, w_{V_W})$. Solving the minimum weighted vertex cover (MWVC) problem on $W(G)$ leads then to a solution for the minimum weighted STC of G, where MWVC is defined as follows. Given a vertex-weighted graph $G = (V, E, w)$, the minimum weighted vertex cover asks if there exists a subset of the vertices $C \subseteq V$ such that each edge $e \in E$ is incident to a vertex $v \in C$ and the sum $\sum_{v \in C} w(v)$ is minimal.

Lemma 1. *Solving the MWVC on $W(G)$ leads to a solution of the minimum weight STC on G.*

Proof. It is known that a (non-weighted) vertex cover $C \subseteq V(W)$ in $W(G)$ is in one-to-one correspondence to a (non-weighted) STC in G, see [39]. The idea is the following. Recall that the wedge graph $W(G)$ contains for each edge $\{i, j\} \in E(G)$ one vertex $n_{ij} \in V(W)$. Two vertices n_{uv}, n_{uw} in $W(G)$ are only adjacent if there exists a wedge $(u, \{v, w\}) \in \mathcal{W}(G)$. If we choose the weak edges $E \setminus S$ to be the edges $\{i, j\} \in E$ such that $n_{ij} \in C$ each wedge has at least one weak edge. Now for the weighted case, by definition, the weight of the STC $\sum_{e \in E \setminus S} w(e)$ equals the weight of a minimum vertex cover $\sum_{v \in C} w_{V_W}(v)$. □

Lemma 1 implies that an approximation for MWVC yields an approximation for WeightedMinSTC. A well-known 2-approximation for the MWVC is the pricing method which we briefly describe in the following. The idea of the pricing algorithm is to assign to each edge $e \in E$ a price $p(e)$ initialized with zero. We say a vertex is *tight* if the sum of the prices of its incident edges equals the weight of the vertex. We iterate over the edges, and if for $e = \{u, v\}$ both u and v are not tight, we increase the price of $p(e)$ until at least one of u or v is tight. Finally, the vertex cover is the set of tight vertices. See, e.g., [22] for a detailed introduction. In Section 5.3, we generalize the pricing algorithm for fully dynamic updates of edge insertions and deletions, and vertex weight updates.

5 Strong Triadic Closure in Temporal Networks

We first present meaningful weighting functions to obtain an edge-weighted aggregated graph from the temporal network. Next, we discuss the approximation of the WeightedMinSTC in the non-streaming case. Finally, we introduce the 2-approximation streaming algorithm for temporal networks.

5.1 Weighting Functions for the Aggregated Graph

A key step in the computation of the STC for temporal networks is the aggregation and weighting of the temporal network to obtain a weighted static network. Recall that the weighting of the aggregated graph $A_\phi(\mathcal{G})$ is determined by the weighting function $\phi : 2^{\mathcal{E}} \to \mathbb{R}$ such that $w(e) = \phi(\mathcal{F}_e)$ with $\mathcal{F}_e = \{e \mid (e, t) \in \mathcal{E}\}$. Naturally, the weighting function ϕ needs to be meaningful in terms of tie strength; hence, we propose the following variants of ϕ.

- *Contact frequency:* We set $\phi(\mathcal{F}_e) = |\mathcal{F}_e|$, i.e., the weight $w(e)$ of edge e in the aggregated graph equals the number of temporal edges between the endpoints of e. Contact frequency is a popular and common substitute for tie strength [10,12,28].
- *Exponential decay:* The authors of [28] proposed to measure tie strength in terms of the recency of contacts. We propose the following weighting variant to capture this property where $\phi(\mathcal{F}_e) = \sum_{i=1}^{|\mathcal{F}_e|-1} e^{-(t(e_{i+1})-t(e_i))}$ if $|\mathcal{F}_e| \geq 2$ and else $\phi(\mathcal{F}_e) = 0$. Here, we interpret $\mathcal{F}_e$ as a chronologically ordered sequence of the edges.
- *Duration:* Temporal networks can include durations as an additional parameter of the temporal edges, i.e., each temporal edge e has an assigned value $\lambda(e) \in \mathbb{N}$ that describes, e.g., the duration of a contact [18]. The duration is also commonly used as an indicator for tie strength [10]. We can define ϕ in terms of the duration, e.g., $\phi(\mathcal{F}_e) = \sum_{f \in \mathcal{F}_e} \lambda(f)$.

Other weighting functions are possible, e.g., combinations of the ones above or weighting functions that include node feature similarities.

5.2 Approximation of WeightedMinSTC

Before introducing our streaming algorithm, we discuss how to compute and approximate the WeightedMinSTC in a temporal network $\mathcal{G} = (V, \mathcal{E})$ in the non-streaming case. Consider the following algorithm:

1. Compute $A_\phi(\mathcal{G}) = (V, E, w)$ using an appropriate weighting function ϕ.
2. Compute the vertex-weighted wedge graph $W(A_\phi(\mathcal{G})) = (V_W, E_W, w_{V_W})$.
3. Compute an MWVC C on $W(A_\phi(\mathcal{G}))$.

The nodes n_{uv} in C then correspond to the weak ties $\{u, v\}$ in $\mathcal{G}$. Depending on how we solve step three, we can either compute an optimal or approximate solution, e.g., using the pricing approximation for the MWVC, we obtain a 2-approximation for WeightedMinSTC. Using the pricing approximation, we have linear running time in the number of edges in the wedge graph. The problem with this direct approach is its limited scalability. The reason is that the number of vertices in the wedge graph $|V_W| = |E(A_\phi(\mathcal{G}))|$ and the number of edges equals the number of wedges in A, which is bounded by $\mathcal{O}(|V|^3)$, see [36], leading to a total running time and space complexity of $\mathcal{O}(|V|^3)$.

5.3 Streaming Algorithm for WeightedMinSTC

In the previous section, we saw that the size of the wedge graph could render the direct approximation approach infeasible for large temporal networks. We use a sliding time window of size $\Delta \in \mathbb{N}$ to compute the changing STC for each time window to overcome this obstacle. The advantage is two-fold: (1) By considering limited time windows, the size of the wedge graphs for which we have to compute the MWVC is reduced because usually not all participants in a network have contact in the same time window. (2) If we consider temporal networks spanning

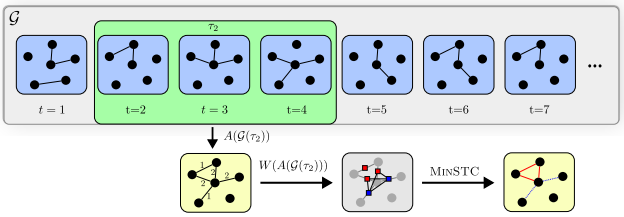

Fig. 2: Example for computing the weighted STC of a sliding time window.

a long (possibly infinite) time range, the relationships, and thus, tie strengths, between participants change over time. Using the sliding time window approach, we are able to capture such changes.

The following discussion assumes the weighting function ϕ to be linear in the contact frequency, and we omit the subscript. But, our results are general and can be applied to other weighting functions. Let τ be a time interval and let $A(\mathcal{G}(\tau))$ be the aggregated graph of $\mathcal{G}(\tau)$, i.e., the temporal network that only contains edges starting and arriving during the interval τ. For a time window size of $\Delta \in \mathbb{N}$, we define the sliding time window τ_t at timestamp t with $t \in [1, T(\mathcal{G}) - \Delta + 1]$ as $\tau_t = [t, t + \Delta - 1]$.

Figure 2 shows an example of our streaming approach for $\Delta = 3$. The first seven timestamps of temporal network $\mathcal{G}$ are shown as static slices. The time window τ_2 of size three starts at $t = 2$. First, the static graph $A(\mathcal{G}(\tau_2))$ is aggregated, and the wedge graph $W(A(\mathcal{G}(\tau_2)))$ is constructed. The wedge graph is used to compute the weighted STC. After this, the time window is moved one time stamp further, i.e., it starts at $t = 3$ and ends at $t = 5$, and the aggregation and STC computation are repeated (not shown in Figure 2). In the following, we describe how the aggregated and wedge graphs are updated when the time window is moved forward, how the MWVC is updated for the changes of the wedge graph, and how the final streaming algorithm proceeds.

Updating the aggregated and wedge graphs Let τ_{t_1} and τ_{t_2} be to consecutive time windows, i.e., $t_2 = t_1 + 1$. Furthermore, let $A_i = A(\mathcal{G}(\tau_{t_i}))$ and $W_i = W(A(\mathcal{G}(\tau_{t_i})))$ with $i \in \{1, 2\}$ be the corresponding aggregated and wedge graphs. The sets of edges appearing in the time windows $\mathcal{G}(\tau_{t_1})$ and $\mathcal{G}(\tau_{t_2})$ might differ. For each temporal edge that is in $\mathcal{G}(\tau_{t_1})$ but not in $\mathcal{G}(\tau_{t_2})$, we reduce the weight of the corresponding edge in the aggregated graph A_1. If the weight reaches zero, we delete the edge from A_1. Analogously, for each temporal edge that is in $\mathcal{G}(\tau_{t_2})$ but not in $\mathcal{G}(\tau_{t_1})$, we increase the weight of the corresponding edge in A_1. If the edge is missing, we insert it. This way, we obtain A_2 from A_1 by a sequence of update operations. Now, we map these edge removals, additions, and edge weight changes between A_1 and A_2 to updates on W_1 to obtain W_2. For each edge removal (addition) $e = \{u, v\}$ between A_1 and A_2, we remove (add) the corresponding vertex (and incident edges) in W_1. We also have to add or remove edges in W_1 depending on newly created or removed wedges. More precisely, for every new wedge in A_1, we add an edge between the corresponding

vertices in W_1, and for each removed wedge, i.e., by deleting an edge or creating a new triangle, in A_1, we remove the edges between the corresponding vertices in W_1. Furthermore, for each edge weight change between A_1 and A_2, we decrease (increase) the weight of the corresponding vertex in W_1. Hence, the wedge graph W_1 is edited by a sequence σ of vertex and edge insertions, vertex and edge removals, and weight changes to obtain W_2. Because we only need to insert or remove a vertex in the wedge graph W_1 if the degree changes between zero and a positive value, we do not consider vertex insertion and removal in W_1 as separate operations in the following. The number of vertices and edges in W_1 is bounded by the current numbers of edges and wedges in A_1. Furthermore, we bound the number of changes in W_1 after inserting or deleting edges from A_1.

Lemma 2. *The number of new edges in W_1 after inserting $\{v, w\}$ into A_1 is at most $d(v) + d(w)$, and the number of edges removed from W is at most $\min(d(v), d(w))$. The number of new edges in W_1 after removing $\{v, w\}$ from A_1 is at most $\min(d(v), d(w))$, and the number of edges removed from W_1 is at most $d(v) + d(w)$.*

Updating the MWVC If the sliding time window moves forward, the current wedge graph W is updated by the sequence σ. We consider the updates occurring one at a time and maintain a 2-approximation of an MWVC in W. Algorithm 1 shows our dynamic pricing approximation based on the non-dynamic 2-approximation for the MWVC. The algorithm supports the needed operations of inserting and deleting edges, as well as increasing and decreasing vertex weights. When called for the first time, an empty vertex cover C and wedge graph W are initialized (line 1f.), which will be maintained and updated in subsequent calls of the algorithm. In the following, we show that our algorithm gives a 2-approximation of the MWVC after each of the update operations.

Definition 2. *We assign to each edge $e \in E(W)$ a price $p(e) \in \mathbb{R}$. We call prices fair, if $s(v) = \sum_{e \in \delta(v)} p(e) \leq w(v)$ for all $v \in V(W)$. And, we say a vertex $v \in V(W)$ is tight if $s(v) = w(v)$.*

Let W be the current wedge graph and σ a sequence of dynamic update requests, i.e., inserting or deleting edges and increasing or decreasing vertex weights in W. Algorithm 1 calls for each request $r \in \sigma$ a corresponding procedure to update W and the current vertex cover C (line 35f.). We show that after each processed request, the following invariant holds.

Invariant 1. *The prices are fair, i.e., $s(v) \leq w(v)$ for all vertices $v \in V(W)$, and $C \subseteq V(W)$ is a vertex cover.*

Lemma 3. *If Invariant 1 holds, after calling one of the procedures* InsEdge, DelEdge, DecWeight, *or* IncWeight *Invariant 1 still holds.*

Theorem 1. *Algorithm 1 maintains a vertex cover with $w(C) \leq 2w(OPT)$.*

Proof. Lemma 3 ensures that C is a vertex cover and after each dynamic update the prices are fair, i.e., $\sum_{e\in\delta(v)} p(e) \leq w(v)$. Furthermore, (1) for an optimal MWVC OPT and fair prices, it holds that $\sum_{e\in E(W)} p(e) \leq w(OPT)$, and (2) for the vertex cover C and the computed prices, it holds that $\frac{1}{2}w(C) \leq \sum_{e\in E(W)} p(e)$. Hence, the result follows. □

We now discuss the running times of the dynamic update procedures. For each of the four operations, the running time is in $\mathcal{O}(F)$, i.e., the size of the set for which we call the UPDATE procedure.

Theorem 2. *Let d_{max} be the largest degree of any vertex in $V(W)$. The running time of* INSEDGE *is in $\mathcal{O}(1)$, and* DELEDGE *is in $\mathcal{O}(d_{max})$.* DECWEIGHT *is in $\mathcal{O}(d_{max}^2)$, and* INCWEIGHT *is in $\mathcal{O}(d_{max})$.*

Algorithm 1: Dynamic Pricing Approximation

Input: Sequence σ of dynamic update requests
Output: Algorithm maintains a 2-approximation of MWVC C

Initialize and maintain vertex cover C
Initialize and maintain wedge graph W

Procedure UPDATE(*set of edges* F):
  foreach $e = \{u, v\} \in F$ **do**
    if *u or v is tight* **then**
      continue
  increase $p(e)$ until u or v tight
  add newly tight vertices to C

Procedure DECWEIGHT(v, w_n):
  $w(v) \leftarrow w_n$
  $C \leftarrow C \setminus \{v\}$
  $F' \leftarrow \delta(v)$
  initialize $F = \emptyset$
  foreach $e = \{v, x\} \in F'$ **do**
    $p(e) \leftarrow 0$
    if *x not tight* **then**
      $C \leftarrow C \setminus \{x\}$
      $F \leftarrow F \cup \{\{x, y\} \in E(G) \mid y \text{ is not tight } \} \cup \{e\}$
  UPDATE(F)

Procedure DELEDGE($e_d = \{u, v\} \in E(W)$):
  $E(W) \leftarrow E(W) \setminus \{e_d\}$
  update $s(u)$ and $s(v)$
  $C \leftarrow C \setminus e_d$
  $F \leftarrow \{\{x, y\} \in E(W) \mid y \in e_d \text{ and } x \text{ is not tight}\}$
  UPDATE(F)

Procedure INSEDGE($e_n = \{u, v\}$):
  $E(W) \leftarrow E(W) \cup \{e_n\}$
  UPDATE($\{e_n\}$)

Procedure INCWEIGHT(v, w_n):
  $w(v) \leftarrow w_n$
  if $v \in C$ **then**
    $C \leftarrow C \setminus \{v\}$
    $F \leftarrow \{\{x, v\} \in E(W) \mid x \text{ is not tight}\}$
    UPDATE(F)

foreach *update request* $r \in \sigma$ **do**
  Call the to r corresponding funct. $f \in \{$INSEDGE, DELEDGE, INCWEIGHT, DECWEIGHT$\}$.

The Streaming Algorithm Algorithm 2 shows the final streaming algorithm that expects as input a stream of chronologically ordered temporal edges

and the time window size Δ. As long as edges are arriving, it iteratively updates the time windows and uses Algorithm 1 to compute the MinWeightSTC approximation for the current time window τ_t with $t \in [1, T(\mathcal{G}) - \Delta]$. Algorithm 2 outputs the strong edges based on the computed vertex cover C_t in line 9. It skips lines 7-9 if there are no changes in E_τ.

Theorem 3. *Let d_t^W (d_t^A) be the maximal degree in W (A, resp.) after iteration t of the while loop in Algorithm 2. The running time of iteration t is in $\mathcal{O}(\xi \cdot d_t^A \cdot (d_t^W)^2)$, with $\xi = \max\{|E_t^-|, |E_t^+|\}$.*

Algorithm 2: Streaming algorithm for the STC in temporal networks

Input: Stream of edges arriving in chronological order, $\Delta \in \mathbb{N}$
Output: 2-Approx. of MinWeightSTC for each time window of size Δ

1 Initialize $t_s = 1$, $t_e = t_s + \Delta - 1$
2 Initialize empty list of edges E_τ and empty aggregated graph A
3 **while** *temporal edges are incoming* **do**
4 Update E_τ for time window $\tau_t = [t_s, t_e]$ such that $\forall e \in E_\tau$ it holds $t(e) \in \tau_t$
5 Let E_τ^- (E_τ^+) be the edges removed from (inserted to) E_τ
6 **if** $E_\tau^- \neq \emptyset$ *or* $E_\tau^+ \neq \emptyset$ **then**
7 Use E_τ^- and E_τ^+ to update A and to obtain the update sequence σ_t
8 Call Algorithm 1 with σ_t to obtain the the MWVC approximation C_t
9 Output $S_t = \{\{u, v\} \in E(A) \mid n_{u,v} \notin C_t\}$
10 Move time window forward by increasing t_s and t_e

6 Experiments

We compare the weighted and unweighted STC on real-world temporal networks and evaluate the efficiency of our streaming algorithm. We use the following algorithms for computing the weighted STC.

- `ExactW` is the weighted exact computation using the ILP (see Appendix A).
- `Pricing` uses the non-dynamic pricing approximation in the wedge graph.
- `DynAppr` is our dynamic streaming Algorithm 2.
- `STCtime` is a baseline streaming algorithm that recomputes the MWVC with the pricing method for each time window.

Moreover, we use the following algorithms for computing the non-weighted STC.

- `ExactNw` is the exact computation using an ILP (see [1]).

- `Matching` is the matching-based approximation of the unweighted vertex cover in the (non-weighted) wedge graph, see [39].
- `HighDeg` is a $\mathcal{O}(\log n)$ approximation by iteratively adding the highest degree vertex to the vertex cover, and removing all incident edges, see [39].

We implemented all algorithms in C++, using GNU CC Compiler 9.3.0 with the flag `--O2` and Gurobi 9.5.0 with Python 3 for solving ILPs. All experiments ran on a workstation with an AMD EPYC 7402P 24-Core Processor with 3.35 GHz and 256 GB of RAM running Ubuntu 18.04.3 LTS, and with a time limit of twelve hours. Our source code is available at `gitlab.com/tgpublic/tgstc`.

Data Sets Table 1 shows the statistics of the real-world data set used for our experiments. Note that for a wedge graph W of an aggregated graph A, $|V(W)| = |E(A)|$, and the number of edges $|E(W)|$ equals the number of wedges in A. For *Reddit* and *StackOverflow* the size of $|E(W)|$ and the number of triangles are estimated using vertex sampling from [43]. Further details of the data sets are available in Appendix B.

Table 1: Statistics of the data sets (*estimated).

Data set	**Properties**							
	$\lvert V\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{T}(\mathcal{G})\rvert$	$\lvert V(W)\rvert$	$\lvert E(W)\rvert$	#Triangles	Domain	Ref.
Malawi	86	102 293	43 438	347	2 254	441	human contact	[34]
Copresence	219	1 283 194	21 536	16 725	549 449	713 002	human contact	[9]
Primary	242	125 773	3 100	8 317	337 504	103 760	human contact	[40]
Enron	87 101	1 147 126	220 312	298 607	45 595 540	1 234 257	communication	[23]
Yahoo	100 001	3 179 718	1 498 868	594 989	18 136 435	590 396	communication	[37]
StackOverflow	2 601 977	63 497 050	41 484 769	28 183 518	*33 898 217 240	*110 670 755	social network	[35]
Reddit	5 279 069	116 029 037	43 067 563	96 659 109	*86 758 743 921	*901 446 625	social network	[15]

6.1 Comparing weighted and non-weighted STC

First, we count the number of strong edges and the mean edge weight of strong edges of the first five data sets. *StackOverflow* and *Reddit* are too large for the direct computation. We use the contact frequency as the weighting function for the aggregated networks. Table 2a shows the percentage of strong edges computed using the different algorithms. The exact computation for *Enron* and *Yahoo* could not be finished within the given time limit. For the remaining data sets, we observe for the exact solutions that the number of strong edges in the non-weighted case is higher than for the weighted case. This is expected, as for edge weights of at least one, the number of strong edges in the non-weighted STC is an upper bound for the number of strong edges in the weighted STC. However, when we look at the quality of the STC by considering how the strong edge weights compare to the empirical strength of the connections, we

Table 2: Comparison of the weighted and non-weighted STC (OOT—out of time)

(a) Percentage of strong edges in aggregated graph.

Data set	Weighted		Non-weighted		
	`ExactW`	`Pricing`	`ExactNw`	`Matching`	`HighDeg`
Malawi	30.83	29.97	37.75	27.38	36.31
Copresence	31.12	21.37	37.95	29.20	35.31
Primary	27.17	21.94	27.83	18.99	27.35
Enron	OOT	2.75	OOT	3.28	4.61
Yahoo	OOT	9.86	OOT	9.98	14.29

(b) Mean edge weights.

Data set	Weighted				Non-weighted					
	`ExactW`		`Pricing`		`ExactNw`		`Matching`		`HighDeg`	
	Weak	Strong	Weak	Strong	Weak	Strong	Weak	Strong	Weak	Strong
Malawi	23.87	902.46	24.40	926.58	218.08	421.27	255.33	399.48	242.84	385.92
Copresence	20.30	78.32	46.13	189.31	27.22	56.56	58.85	120.07	57.13	112.63
Primary	2.73	20.48	6.58	45.50	3.34	18.49	9.32	39.88	6.19	38.84
Enron	OOT	OOT	3.69	9.33	OOT	OOT	3.77	6.01	3.76	5.50
Yahoo	OOT	OOT	4.37	14.23	OOT	OOT	4.78	10.42	4.60	9.84

see the benefits of our new approach. An STC labeling with strong edges with high average weights and weak edges with low average weights is favorable. The mean weights of the strong and weak edges are shown in Table 2b. `Pricing` leads to the highest mean edge weight for strong edges in almost all data sets. The mean weight of the strong edges for the exact methods is always significantly higher for `ExactW` than `ExactNw`. The reason is that `ExactNw` does not consider the edge weights. Furthermore, it shows the effectiveness of our approach and indicates that the empirical a priori knowledge given by the edge weights is successfully captured by the weighted STC. To further verify this claim, we evaluated how many of the highest-weight edges are classified as strong. To this end, we computed the *precision* and *recall* for the top-100 weighted edges in the aggregated graph and the set of strong edges. Let H be the set of edges with the top-100 highest degrees. The precision is defined as $p = |H \cap S|/|S|$ and the recall as $r = |H \cap S|/|H|$. Figure 3 shows the results. Note that the y-axis of precision uses a logarithmic scale. The results show that the algorithms for the weighted STC lead to higher precision and recall values for all data sets.

6.2 Efficiency of the streaming algorithm

In order to evaluate our streaming algorithm, we measured the running times on the *Enron, Yahoo, StackOverflow*, and *Reddit* data sets with time window sizes Δ

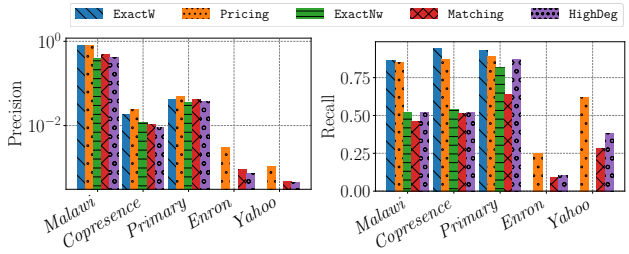

Fig. 3: Precision and recall for classifying the top-100 highest weighted edges in the aggregated graph as strong edges. The y-axis of precision is logarithmic.

Table 3: Running times in seconds of the streaming alg. (OOT—out of time).

Data set	$\Delta = 1$ hour		$\Delta = 1$ day		$\Delta = 1$ week	
	`DynAppr`	`STCtime`	`DynAppr`	`STCtime`	`DynAppr`	`STCtime`
Enron	264.74	**89.18**	**306.13**	1 606.09	**352.01**	20 870.77
Yahoo	**15.99**	767.40	**91.46**	OOT	**144.52**	OOT
StackOverflow	**170.38**	2 298.58	**971.22**	OOT	**16 461.53**	OOT
Reddit	**1 254.66**	13 244.84	**37 627.79**	OOT	OOT	OOT

of one hour, one day, and one week, respectively. Table 3 shows the results. In almost all cases, our streaming algorithm `DynAppr` beats the baseline `STCtime` with running times that are often orders of magnitudes faster. The reason is that `STCtime` uses the non-dynamic pricing approximation, which needs to consider all edges of the current wedge graph in each time window. Hence, the baseline is often not able to finish the computations within the given time limit, i.e., for seven of the twelve experiments, it runs out of time. The only case in which the baseline is faster than `DynAppr` is for the *Enron* data set and a time window size of one hour. Here, the computed wedge graphs of the time windows are, on average, very small (see Figure 2 (a) in Appendix C.2), and the dynamic algorithm can not make up for its additional complexity due to calling Algorithm 1. However, we also see for *Enron* that for larger time windows, the running times of the baseline strongly increase, and for `DynAppr`, the increase is slight. In general, the number of vertices and edges in the wedge graphs increases with larger time window sizes Δ (see Figure 2 in Appendix C.2). Hence, the running times increase for both algorithms with increasing Δ. In the case of *Reddit* and a time window size of one week, the sizes of the wedge graphs are too large to compute all solutions within the time limit, even for DynAppr.

7 Conclusion and Future Work

We generalized the STC to a weighted version to include a priori knowledge in the form of edge weights representing empirical tie strength. We applied our

new STC variant to temporal networks and showed that we obtained meaningful results. Our main contribution is our 2-approximative streaming algorithm for the weighted STC in temporal networks. We empirically validated its efficiency in our evaluation. Furthermore, we introduced a fully dynamic 2-approximation of the MWVC problem with efficient update routines as part of our streaming algorithm. It might be of interest in itself.

As an extension of this work, discussion of further variants of the STC can be interesting. For example, [39] introduced variants with edge additions and multiple relationship types. Efficient streaming algorithms for weighted versions of these variants are planned as future work.

References

1. Adriaens, F., De Bie, T., Gionis, A., Lijffijt, J., Matakos, A., Rozenshtein, P.: Relaxing the strong triadic closure problem for edge strength inference. Data Mining and Knowledge Discovery pp. 1–41 (2020)
2. Ahmadian, S., Haddadan, S.: A theoretical analysis of graph evolution caused by triadic closure and algorithmic implications. In: International Conference on Big Data (Big Data). pp. 5–14. IEEE (2020)
3. Bhattacharya, S., Henzinger, M., Italiano, G.F.: Deterministic fully dynamic data structures for vertex cover and matching. J. on Computing **47**(3), 859–887 (2018)
4. Candia, J., González, M.C., Wang, P., Schoenharl, T., Madey, G., Barabási, A.L.: Uncovering individual and collective human dynamics from mobile phone records. Journal of physics A: mathematical and theoretical **41**(22), 224015 (2008)
5. Chen, J., Molter, H., Sorge, M., Suchý, O.: Cluster editing in multi-layer and temporal graphs. In: International Symposium on Algorithms and Computation, ISAAC. LIPIcs, vol. 123, pp. 24:1–24:13. Schloss Dagstuhl–LZI (2018)
6. Easley, D.A., Kleinberg, J.M.: Networks, Crowds, and Markets - Reasoning About a Highly Connected World. Cambridge University Press (2010)
7. Eckmann, J.P., Moses, E., Sergi, D.: Entropy of dialogues creates coherent structures in e-mail traffic. Proc. of the Nat. Acad. of Sc. **101**(40), 14333–14337 (2004)
8. Gallai, T.: Transitiv orientierbare graphen. Acta M. Hung. **18**(1-2), 25–66 (1967)
9. Génois, M., Barrat, A.: Can co-location be used as a proxy for face-to-face contacts? EPJ Data Science **7**(1), 11 (2018)
10. Gilbert, E., Karahalios, K.: Predicting tie strength with social media. In: Proceedings of the 27th International Conference on Human Factors in Computing Systems, CHI. pp. 211–220. ACM (2009)
11. Gilbert, E., Karahalios, K., Sandvig, C.: The network in the garden: an empirical analysis of social media in rural life. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 1603–1612 (2008)
12. Granovetter, M.S.: The strength of weak ties. A. J. of Soc. **78**(6), 1360–1380 (1973)
13. Grüttemeier, N., Komusiewicz, C.: On the relation of strong triadic closure and cluster deletion. Algorithmica **82**, 853–880 (2020)
14. Hanneke, S., Xing, E.P.: Discrete temporal models of social networks. In: ICML Workshop on Statistical Network Analysis. pp. 115–125. Springer (2006)
15. Hessel, J., Tan, C., Lee, L.: Science, askscience, and badscience: On the coexistence of highly related communities. In: Proceedings of the International AAAI Conference on Web and Social Media. vol. 10, pp. 171–180 (2016)

16. Himmel, A., Molter, H., Niedermeier, R., Sorge, M.: Adapting the bron-kerbosch algorithm for enumerating maximal cliques in temporal graphs. Soc. Netw. Anal. Min. **7**(1), 35:1–35:16 (2017)
17. Holme, P., Edling, C.R., Liljeros, F.: Structure and time evolution of an internet dating community. Social Networks **26**(2), 155–174 (2004)
18. Holme, P., Saramäki, J.: Temporal networks. Physics reports **519**(3), 97–125 (2012)
19. Huang, H., Tang, J., Wu, S., Liu, L., Fu, X.: Mining triadic closure patterns in social networks. In: Intl. Conf. on World Wide Web, WWW. pp. 499–504. ACM (2014)
20. Kahanda, I., Neville, J.: Using transactional information to predict link strength in online social networks. In: Proceedings of the International AAAI Conference on Web and Social Media. vol. 3, pp. 74–81 (2009)
21. Kempe, D., Kleinberg, J.M., Kumar, A.: Connectivity and inference problems for temporal networks. J. Comput. Syst. Sci. **64**(4), 820–842 (2002)
22. Kleinberg, J., Tardos, E.: Algorithm design. Pearson Education India (2006)
23. Klimt, B., Yang, Y.: The Enron corpus: A new dataset for email classification research. In: European Conf. on Machine Learning. pp. 217–226. Springer (2004)
24. Konstantinidis, A.L., Nikolopoulos, S.D., Papadopoulos, C.: Strong triadic closure in cographs and graphs of low maximum degree. Th. Comp. Sci. **740**, 76–84 (2018)
25. Konstantinidis, A.L., Papadopoulos, C.: Maximizing the strong triadic closure in split graphs and proper interval graphs. Discret. Appl. Math. **285**, 79–95 (2020)
26. Kossinets, G., Watts, D.J.: Empirical analysis of an evolving social network. science **311**(5757), 88–90 (2006)
27. Latapy, M., Viard, T., Magnien, C.: Stream graphs and link streams for the modeling of interactions over time. Soc. Netw. Anal. Min. **8**(1), 61:1–61:29 (2018)
28. Lin, N., Dayton, P.W., Greenwald, P.: Analyzing the instrumental use of relations in the context of social structure. Sociol. Meth. & Research **7**(2), 149–166 (1978)
29. Matakos, A., Gionis, A.: Strengthening ties towards a highly-connected world. Data Min. Knowl. Discov. **36**(1), 448–476 (2022)
30. Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Measurement and analysis of online social networks. In: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement. pp. 29–42 (2007)
31. Moinet, A., Starnini, M., Pastor-Satorras, R.: Burstiness and aging in social temporal networks. Physical review letters **114**(10), 108701 (2015)
32. Oettershagen, L., Konstantinidis, A.L., Italiano, G.F.: Inferring tie strength in temporal networks (2022), `https://arxiv.org/abs/2206.11705`
33. Oettershagen, L., Kriege, N.M., Morris, C., Mutzel, P.: Classifying dissemination processes in temporal graphs. Big Data **8**(5), 363–378 (2020)
34. Ozella, L., Paolotti, D., Lichand, G., Rodríguez, J.P., Haenni, S., Phuka, J., Leal-Neto, O.B., Cattuto, C.: Using wearable proximity sensors to characterize social contact patterns in a village of rural malawi. EPJ Data Science **10**(1), 46 (2021)
35. Paranjape, A., Benson, A.R., Leskovec, J.: Motifs in temporal networks. In: Proc. of the tenth ACM intl. conf. on web search and data mining. pp. 601–610 (2017)
36. Pyatkin, A., Lykhovyd, E., Butenko, S.: The maximum number of induced open triangles in graphs of a given order. Optimization Letters **13**(8), 1927–1935 (2019)
37. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: AAAI (2015), `https://networkrepository.com`
38. Rozenshtein, P., Tatti, N., Gionis, A.: Inferring the strength of social ties: A community-driven approach. In: Proc. of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1017–1025. ACM (2017)

39. Sintos, S., Tsaparas, P.: Using strong triadic closure to characterize ties in social networks. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1466–1475 (2014)
40. Stehlé, J., Voirin, N., Barrat, A., Cattuto, C., Isella, L., Pinton, J.F., Quaggiotto, M., Van den Broeck, W., Régis, C., Lina, B., et al.: High-resolution measurements of face-to-face contact patterns in a primary school. PloS one **6**(8), e23176 (2011)
41. Tantipathananandh, C., Berger-Wolf, T.Y.: Finding communities in dynamic social networks. In: Intl. Conference on Data Mining, ICDM. pp. 1236–1241. IEEE (2011)
42. Wei, H.T., Hon, W.K., Horn, P., Liao, C.S., Sadakane, K.: An O(1)-Approximation Algorithm for Dynamic Weighted Vertex Cover with Soft Capacity. In: Approx., Random., and Combinatorial Opt. Algor. and Techniques (APPROX/RANDOM 2018). LIPIcs, vol. 116, pp. 27:1–27:14. Schloss Dagstuhl–LZI (2018)
43. Wu, B., Yi, K., Li, Z.: Counting triangles in large graphs by random sampling. IEEE Transactions on Knowledge and Data Engineering **28**(8), 2013–2026 (2016)
44. Xiang, R., Neville, J., Rogati, M.: Modeling relationship strength in online social networks. In: Intl. Conf. on World Wide Web, WWW. pp. 981–990. ACM (2010)
45. Zhou, L., Yang, Y., Ren, X., Wu, F., Zhuang, Y.: Dynamic network embedding by modeling triadic closure process. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence. pp. 571–578. AAAI Press (2018)