Data Mining and Knowledge Discovery (2025) 39:16
https://doi.org/10.1007/s10618-025-01093-8

Inferring tie strength in temporal networks

Lutz Oettershagen[1] · Athanasios L. Konstantinidis[2] · Giuseppe F. Italiano[3]

Received: 9 January 2023 / Accepted: 5 February 2025 / Published online: 1 March 2025

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. A variant of the STC called STC+ allows adding a few new weak edges to obtain improved solutions. So far, most works discuss the STC or 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. Similarly, we introduce a generalized weighted version of the STC+. The weighted STC is hard to compute, and our main contribution is an efficient 2-approximation (resp. 3-approximation) streaming algorithm for the weighted STC (resp. STC+) in temporal networks. As a technical contribution, we introduce a fully dynamic k-approximation for the minimum weighted vertex cover problem in hypergraphs with edges of size k, which is a crucial component of our streaming algorithms. An empirical evaluation shows that the weighted STC leads to solutions that better capture the a priori knowledge given by the edge weights than the non-weighted STC. Moreover, we show that our streaming algorithm efficiently approximates the weighted STC in real-world large-scale social networks.

Keywords Triadic closure · Temporal network · Tie strength inference

Responsible editor: Johannes Fürnkranz.

A preliminary version of this paper was presented at the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2022 (ECMLPKDD 2022) (Oettershagen et al. 2022). Partially supported by MUR, the Italian Ministry for University and Research, under PRIN Project AHeAD (efficient Algorithms for HArnessing networked Data).

Extended author information available on the last page of the article

(a) Example for optimal weighted STC. The edge weights correspond to the number 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

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 (Gilbert et al. 2008; Kahanda and Neville 2009). Users of large-scale social networks are commonly connected to hundreds or even thousands of other participants (Kossinets and Watts 2006; Mislove et al. 2007). 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 (1973), the topic of tie strength inference has 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 (Gilbert and Karahalios 2009; Kahanda and Neville 2009; Rozenshtein et al. 2017). A recent work by Sintos and Tsaparas (2014) 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.

We first generalize the ideas of Sintos and Tsaparas (2014) 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 we expect to be 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 Fig. 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 Fig. 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.

Sintos and Tsaparas (2014) also introduced a variant of the STC called STC+ that allows adding new weak edges to obtain improved solutions. Similarly to the standard variant, we introduce a weighted version of the STC+.

We employ these generalizations of the STC and the STC+ to infer 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 (Holme and Saramäki 2012; Rozenshtein and Gionis 2019; Gionis et al. 2024). Temporal networks can naturally be used as models for real-life scenarios, e.g., communication (Candia et al. 2008; Eckmann et al. 2004), contact (Ciaperoni et al. 2020; Oettershagen et al. 2020), and social networks (Hanneke and Xing 2006; Holme et al. 2004; Moinet et al. 2015). 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 (Holme and Saramäki 2012). 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 or STC+, respecting the a priori information given by the edge weights.

A major challenge is that the weighted STC and STC+ are 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-approximation of the minimum weighted STC, i.e., the problem that asks for the minimum number of weak edges. In the case of the STC+, our streaming algorithm efficiently updates a 3-approximation of the minimum weighted STC+.

Note that we infer the tie strength based on the topology of the network. There are various studies on predicting the strength of ties given other features of a network. Gilbert and Karahalios (2009) developed a predictive model to characterize ties in social networks as strong or weak with high accuracy by taking user similarities and interactions into account. In the same direction, Xiang et al. (2010) gave an unsupervised model to infer relationship strength based on user similarity and interaction activity. Moreover, Pham et al. (2016) showed that spatio-temporal features of social interactions could increase the accuracy of inferred ties strength. 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 (1973). An extensive analysis of the STC can be found in the book of Easley and Kleinberg (2010). Sintos and Tsaparas (2014) not only introduced the optimization problem by characterizing the edges of the network as strong or weak using only the structure of the network, but they also proved that the problem of maximizing the strong edges is NP-hard, and provided two approximation algorithms to solve the dual problem of minimizing the weak edges. In the following works, the authors of Grüttemeier and Komusiewicz (2020); Konstantinidis et al. (2018); Konstantinidis and

Papadopoulos (2020) focused on restricted networks to further explore the complexity of STC maximization. Rozenshtein et al. (2017) discuss the STC with additional community connectivity constraints. Adriaens et al. (2020) proposed integer linear programming formulations and corresponding relaxations. Very recently, Matakos and Gionis (2022) proposed a new problem that uses the strong ties of the network to add new edges and increase its connectivity. Veldt (2022) presented connections between the cluster editing problem and the STC+. In the cluster editing problem, given a undirected graph G, the task is to find the minimum number of edges that need to be introduced to or deleted from G to obtain a disjoint union of cliques. The author showed that an α-approximation for STC+ leads to a 2α-approximation for the cluster editing problem.

The mentioned works only consider static networks and do not include edge weights in the computation of the STC or STC+. We propose weighted variants and use them to infer ties strength in temporal networks.

Even though temporal networks are a quite recent research field, there are some comprehensive surveys that introduce the notation, terminology, and applications (Holme and Saramäki 2012; Latapy et al. 2018; Michail 2016). Additionally, there are systematic studies into the complexity of well-known graph problems on temporal networks (e.g. Himmel et al. (2017); Kempe et al. (2002); Viard et al. (2016)). The problem of finding communities and clusters, which can be considered as a related problem, has been studied on temporal networks (Chen et al. 2018; Tantipathananandh and Berger-Wolf 2011). Furthermore, Zhou et al. (2018) 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. (2014) studied the formation of closed triads in dynamic networks. The authors of Ahmadian and Haddadan (2020) introduce a probabilistic model for dynamic graphs based on the triadic closure.

Finally, Wei et al. (2018) 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 (Bhattacharya et al. 2018). However, the algorithm does not support updates of the vertex weights, which is an essential operation in our streaming algorithm.

Our Contributions:

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. Similarly, we generalize the STC+ variant for weighted graphs that allows insertions of new weak edges and extend the concept to temporal graphs.
2. We provide a time-window streaming algorithm framework to efficiently approximate the weighted STC and STC+ over time with an approximation factor of two and three, respectively. As a technical contribution, we propose an efficient dynamic k-approximation for the minimum weighted vertex cover problem (MWVC) in k-uniform hypergraphs, a key ingredient of our streaming framework.
3. Our evaluation using real-world temporal networks shows that the weighted STC and STC+ lead 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 while keeping the same solution quality.

In contrast to the conference version (Oettershagen et al. 2022), this work contains all previously omitted details and proofs. Furthermore, we include the following major additions:

1. **Weighted STC+:** We additionally introduce the weighted STC+ problem and extend our discussion and solution for it.
2. **Dynamic k-Approximation for MWVC:** We lifted the fully dynamic 2-approximation of the minimum weight vertex cover to a fully dynamic k-approximation in k-uniform hypergraphs.
3. **Extended experiments:** We provide additional experiments, including the evaluation of the newly introduced weighted STC+.

The remainder of this paper is organized as follows. In Sect. 2, we introduce the preliminaries and definitions. Section 3 presents the generalization of the STC and the STC+ for weighted networks. Next, in Sect. 4, we discuss the application of the weighted STC and STC+ for tie strength inference in temporal networks. Furthermore, we introduce our new streaming algorithm, including our dynamic minimum weight vertex cover algorithm. In Sect. 5, we evaluate our new approaches, and finally, in Sect. 6, we provide some concluding remarks and discuss possible directions for future work.

2 Preliminaries

Table 1 gives an overview of the used notation and symbols. An undirected *hypergraph* $H = (V, E)$ consists of a finite set of nodes V and a finite set of *hyperedges* $E \subseteq 2^V \setminus \emptyset$, i.e., each hyperedge connects a non-empty subset of V. A hypergraph with all its hyperedges of size k is called k-uniform hypergraph. In particular, we consider the two special cases $k \in \{2, 3\}$. In the first case, the hypergraph is a conventional undirected graph which we may denote with G and for which we call the hyperedges just edges. In the second case, each hyperedge is an unordered triple. We use $V(H)$ and $E(H)$ to denote the sets of nodes and hyperedges, respectively, of H. The set $\delta(v) = \{e \mid e \in E(H), v \in e\}$ contains all hyperedges incident to $v \in V(H)$, and we use $d(v) = |\delta(v)|$ to denote the degree of $v \in V$. An *edge-weighted* undirected hypergraph $H = (V, E, w_E)$ is an undirected hypergraph with additional weight function $w_E : E \to \mathbb{R}$. Analogously, we define a *vertex-weighted* undirected hypergraph $H = (V, E, w_V)$ with a weight function for the vertices $w_V : V \to \mathbb{R}$. If the context is clear, we omit the subscript of the weight function.

For 2-uniform hypergraphs, i.e., undirected graphs, we define *wedges*. A *wedge* is defined as a triplet of nodes $u, v, w \in V$ such that $\{\{u, v\}, \{v, w\}\} \subseteq E$

Table 1 Commonly used notations

Symbol	Definition
$H = (V, E)$	hypergraph
$G = (V, E)$	static undirected graph
V, $V(G)$	finite set of nodes, nodes of graph G
E, $E(G)$	finite sets of (hyper-)edges, (hyper-)edges of G
$\{u, v\} \in E$	static edge
$n = \|V\|$	number of nodes
$m = \|E\|$	number of (hyper-)edges
$\delta(v) = \{e \mid e \in E(H), v \in e\}$	set of all hyperedges incident to $v \in V(H)$
$d(v) = \|\delta(v)\|$	degree of node v
$w_E : E \to \mathbb{R}$	edge weight function
$w_V : V \to \mathbb{R}$	vertex weight function
$(v, \{u, w\})$	wedge (or open triangle) with edge $\{u, w\}$ missing
$\mathcal{W}(G)$	set of wedges in graph G
$W(G) = (U, H, w_V)$	wedge graph of G
$U = \{n_{uv} \mid \{u, v\} \in E\}$	nodes in wedge graph
$H = \{\{n_{uv}, n_{vw}\} \mid (v, \{u, w\}) \in \mathcal{W}(G)\}$	edges in wedge graph
n_{uv}	node in $W(G)$ corresponding to edge $\{u, v\}$ in G
$\mathcal{G} = (V, \mathcal{E})$	temporal graph with temporal edges $\mathcal{E}$
$(\{u, v\}, t) \in \mathcal{E}$	temporal edge with time stamp t
$t(e)$	time stamp of temporal edge e
$A_\phi(\mathcal{G})$	aggregated graph
$\phi : 2^{\mathcal{E}} \to \mathbb{R}$	weighting function for temporal edge sets
$T(\mathcal{G})$	time interval spanned by temporal network $\mathcal{G}$
$S \subseteq E$	set of strong labeled edges
$\alpha \in \mathbb{R}$	weighting factor
$p(e)$	price of edge e
C	vertex cover
$w(C)$	weight of vertex cover
τ	time interval
$\mathcal{G}(\tau)$	temporal subgraph with only edges in interval τ
$\Delta \in \mathbb{N}$	time window size
σ	sequence of dynamic update requests

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 (Gallai 1967).

Definition 1 Let $G = (V, E, w_E)$ be an 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* $e = (\{u, v\}, t, \lambda)$ with u and v in V, $u \neq v$, and *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 timestamp 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 (see Sect. 4.1 for details). 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 Sect. 4.1. 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 specify the labeling unambiguously by the subset of strong edges $S \subseteq E$. Each edge $e \in S$ is called *strong*, and $e \in E \setminus S$ *weak*. We refer to the set S as a *strong-weak labeling*. 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 S *fulfills* the strong triadic closure. In other words, in a strong triadic closure, there is no pair of strong edges $\{u, v\} \in S$ and $\{v, w\} \in S$ 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 (Sintos and Tsaparas 2014). 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 $|S| \geq k$?

Equivalently, we can define the problem based on the weak edges, MinSTC, in which given a graph $G = (V, E)$ and a non-negative integer ℓ. Does there exist $E' \subseteq E$ that $E \setminus E'$ fulfills the strong triadic closure and $|E'| \leq \ell$?

Strong Triadic Closure with Edge Additions Here apart from labeling the edges of the graph as strong or weak, we can add new edges between non-adjacent nodes and label these edges as weak. We denote this problem by MinSTC+ and it is stated as follows: Given a graph $G = (V, E)$ and a non-negative integer ℓ. Does there exist a set $F \subseteq \binom{V}{2} \setminus E$ and a set of weak edges $W \subseteq E$ such that the labeling $S = E \setminus W$ fulfills the strong triadic closure in $G' = (V, E \cup F)$ and $|W \cup F| \leq \ell$?

The motivation for considering edge additions is the following (Sintos and Tsaparas 2014): Let G be an (almost) complete graph on n vertices with exactly one edge $\{u, v\}$ missing. In this case, the best strong-weak labeling for G contains $n - 2$ weak edges. By adding the single edge $\{u, v\}$ to G, we obtain the complete graph over n vertices for which all but $\{u, v\}$ can be labeled strong. Thus, by adding a single edge, we achieve a significant improvement of the labeling.

Finally, note that adding only weak ties to address STC violations mirrors the realistic formation of relationships, which often begin as weak ties. Strong ties,

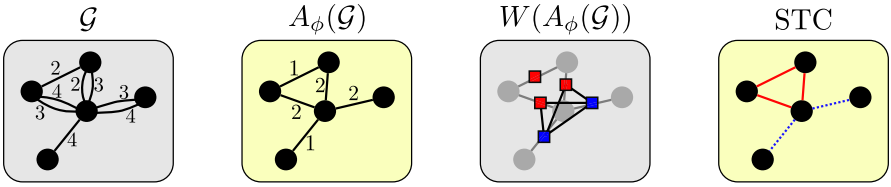


Fig. 2 Example for temporal graph $\mathcal{G}$, the aggregated graph $A_\phi(\mathcal{G})$, the wedge graph $W(A_\phi(\mathcal{G}))$, and minimum weight STC

requiring trust, effort, and a shared history, should not be artificially imposed. In contrast, new weak ties are considered plausible additions based on the existing network topology.

2.1 Examples of temporal, aggregated, and wedge graph

Figure 2 shows from left to right (i) a temporal graph $\mathcal{G}$ and (ii) its aggregated graph $A_\phi(\mathcal{G})$ with ϕ being the weighting function according to contact frequency. Furthermore, (iii) the wedge graph $W(A(\mathcal{G}))$ with highlighted minimum weight vertex cover. The blue nodes are the vertices in the cover. Finally, (iv) the corresponding minimum weight STC labeling of the edges in $A(\mathcal{G})$ is shown. The red edges are strong, and the blue dashed ones are weak.

3 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 into account the weights of the edges by the importance given by w. 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 (2014) 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[1]. Here, we search a strong-weak labeling that fulfills the STC and minimizes

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

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 Sect. 3.2. The advantage of WeightedMinSTC is that we can obtain a 2-approximation.

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 Sintos and Tsaparas (2014). 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 $E \backslash S$ 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 $\sum_{e \in E \backslash S} w(e)$ 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 is based on the primal-dual method and was first suggested in Bar-Yehuda and Even (1981). 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., Kleinberg and Tardos (2006) for a detailed introduction. In Sect. 4.3, we generalize the pricing algorithm for fully dynamic updates of edge insertions and deletions and vertex weight updates.

3.1 Weighted STC+ variant

As above, let $G = (V, E, w)$ be a graph with edge weights reflecting the importance of the edges. We consider the weighted STC+ variant that allows insertions of weak edges called WeightedMinSTC+. Our goal is to find a subset of new edges $E_N \subseteq \binom{V}{2} \setminus E$ and a strong-weak labeling S such that (1) S fulfills the STC in $G = (V, E \cup E_N)$ and (2) the weight $\sum_{e \in E_N \cup (E \backslash S)} w(e)$ is minimized. The new edges in E_N are considered to be weak; thus, no new open wedges that violate strong triadic closure are introduced. We still need to assign the weight $w(f)$

to the new edges $f \in E_N$. Each edge $f = \{u, w\}$ can be part of a set of wedges $\mathcal{W}_f = \{(v, \{u, w\}) \mid v \in V\} \subseteq \mathcal{W}(G)$. We set the weight of $f = \{u, w\}$ to

$$w(f) = \alpha \cdot \sum_{(v,\{u,w\}) \in \mathcal{W}_f} \frac{w(\{v,u\}) + w(\{v,w\})}{|\mathcal{W}_f|},$$

with the parameter $\alpha \in \mathbb{R}_{>0}$, which can be used to allow more or fewer new edges. Note that for unit weights, i.e., in the unweighted STC+ case, we obtain the weight of $w(f) = 1$ using $\alpha = \frac{1}{2}$. Furthermore, for a large enough α, e.g., $\alpha \geq \sum_{(v,\{u,w\}) \in \mathcal{W}} (w(\{v,u\}) + w(\{v,w\}))$, the the solution of the weighted STC+ does not contain any edge $f \in E_N$ and is a solution for the WeightedMinSTC problem.

Similarly to WeightedMinSTC, we provide an approximation based on the minimum vertex cover problem. To this end, we construct a *vertex-weighted hypergraph* $H_W = (V_W, E_W, w_{V_W})$ that contains a hyperedge for each wedge in $G = (V, E, w_E)$. Particularly, we define

$$V_W = \{n_{uv} \mid \{u, v\} \in E\} \cup \{n'_{xy} \mid \{x, y\} \in E_N \subseteq \tbinom{V}{2} \setminus E\}, \text{ and}$$
$$E_W = \{\{n_{uv}, n_{vw}, n'_{uw}\} \mid (v, \{u, w\}) \in \mathcal{W}(G)\}.$$

and the vertex weight function $w_V(n_{uv}) = w(\{u, v\})$. Now, a minimum weight vertex cover in H_W leads to an optimal solution of WeightedMinSTC+. Let $C \subseteq V_W$ be a vertex cover in H_W, i.e., for each hyperedge $e \in E_W$, there exists a vertex $n_{uv} \in e$ with n_{uv} also in C, and the corresponding edge $\{u, v\} \in E \cup E_N$ is labeled weak. Consequently, in case $\{u, v\} \in E$, the to e corresponding wedge has a weak edge. And if $\{u, v\} \in E_N$, the to e corresponding wedge is closed by the new edge $\{u, v\}$. Hence, C leads to a valid labeling for G. Because C has minimum weight, the corresponding labeling is optimal.

We can obtain a 3-approximation algorithm using the pricing method analogously to the approximation for WeightedMinSTC. Note that this improves the $\mathcal{O}(\log n)$ approximation stated in Sintos and Tsaparas (2014).

Theorem 1 *Solving the MWVC in the* 3*-uniform hypergraph* H_W *gives us a* 3*-approximation for the WeightedMinSTC+ problem.*

3.2 ILP formulations

While the weighted STC and STC+ problems can be naturally formulated as an Integer Linear Program (ILP) by extending the unweighted version (e.g., (Adriaens et al. 2020; Veldt 2022)), we include the formulations here for completeness. Let w_{ij} be the weight of edge $\{i, j\}$. We define the ILP for MinWeightSTC, where we use binary variables y_{ij} which encode the if edge $\{i, j\} \in E$ is strong ($y_{ij} = 0$) or weak ($y_{ij} = 1$), as

$$\min \sum_{ij \in E} w_{ij} y_{ij} \tag{1}$$

$$\text{s.t.} \quad y_{ij} + y_{ik} \geq 1 \quad \text{for all } (i, \{j,k\}) \in \mathcal{W}(G) \tag{2}$$

$$y_{ij} \in \{0,1\} \quad \text{for all } \{i,j\} \in E. \tag{3}$$

In the case of the WeightedMinSTC+ problem, we adapt the ILP formulation stated for the unweighted version of the minimum STC+ problem (Veldt 2022). We use binary variables y_{ij} which encode the if (possibly new) edge $\{i,j\} \in E \cup E_N$ is strong ($y_{ij} = 0$) or weak ($y_{ij} = 1$). Hence, for WeightedMinSTC+, we have

$$\min \sum_{ij \in \binom{V}{2}} w_{ij} y_{ij} \tag{4}$$

$$\text{s.t.} \quad y_{ij} + y_{ik} + y_{jk} \geq 1 \quad \text{for all } (i, \{j,k\}) \in \mathcal{W}(G) \tag{5}$$

$$y_{ij} \in \{0,1\} \quad \text{for all } \{i,j\} \in \binom{V}{2}. \tag{6}$$

4 Strong triadic closure in temporal networks

In this section, we present a framework for extending our methods to temporal networks to compute tie strengths between nodes. Specifically, we introduce the TempMinSTC and TempMinSTC+ problems, which are adaptations of the STC and STC+ problems to the temporal setting. Given a temporal network $\mathcal{G}$, the objective is to infer the tie strength between pairs of nodes that have contacts over time within the network.

We first present meaningful weighting functions to obtain an edge-weighted aggregated graph from the temporal network. Next, we discuss the direct approximation of the TempMinSTC and the TempMinSTC+ and its downsides. Finally, to overcome the limitations of the direct computation, we introduce the approximation streaming algorithms for temporal networks.

4.1 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:

- *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 (Gilbert and Karahalios 2009; Granovetter 1973; Lin et al. 1978).
- *Exponential decay:* Lin et al. (1978) 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 (Holme and Saramäki 2012). The duration is also commonly used as an indicator for tie strength (Gilbert and Karahalios 2009). We can define ϕ in terms of the duration, e.g., $\phi(\mathcal{F}_e) = \sum_{f \in \mathcal{F}_e} \lambda(f)$.

Note that these weighting functions are reasonable examples that capture different temporal aspects of the network, but other weighting functions are possible, e.g., combinations of the ones above or weighting functions that include node feature similarities.

4.2 Direct approximation of TempMinSTC and TempMinSTC+

Before introducing our streaming algorithm, we discuss how to compute and approximate the TempMinSTC and TempMinSTC+ in a temporal network $\mathcal{G} = (V, \mathcal{E})$ in the non-streaming case. Algorithm 1 shows the computation for TempMinSTC. The idea is to first aggregate the given temporal graph $\mathcal{G}$ to $A(\mathcal{G})$ and then solve the WeightedMinSTC problem on the aggregated graph.

Algorithm 1 TempMinSTC algorithm for temporal networks

Input: Temporal graph $\mathcal{G}$
Output: Labeling of the ties S
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 $W(A_\phi(\mathcal{G}))$.
4 Return labeling corresponding to C, i.e., nodes $n_{uv} \in C$ then correspond to the weak ties $\{u, v\}$ in $\mathcal{G}$

For the TempMinSTC+, we can compute in step 2 the vertex-weighted wedge hypergraph as described in Sect. 3.1, i.e. solve the WeightedMinSTC+ problem on $A(\mathcal{G})$. 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 TempMinSTC or a 3-approximation for TempMinSTC+, respectively. Using the pricing approximation, we have a linear running time in the number of (hyper-)edges in the wedge graph. The problem with this direct approach of computing a solution for the TempMinSTC or TempMinSTC+ problem is its limited scalability. The reason is that the number of vertices in the wedge

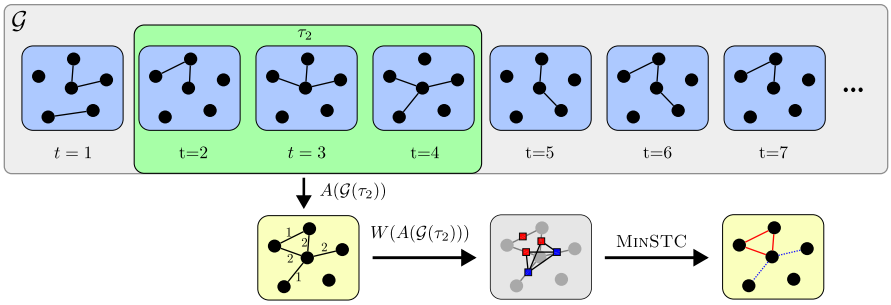


Fig. 3 Example for computing the weighted STC of a sliding time window

graph $|V_W| \geq |E(A_\phi(\mathcal{G}))|$ and the number of edges equals the number of wedges in A, which is bounded by $\mathcal{O}(|V|^3)$, see Pyatkin et al. (2019), leading to the following result.

Theorem 2 *The asymptotic running time and space complexity of Algorithm* 1 *is in* $\mathcal{O}(|V|^3)$.

4.3 Streaming framework for TempMinSTC and TempMinSTC+

In the previous section, we saw that the size of the wedge graph could render the direct approximation approach infeasible for large temporal networks. To overcome this obstacle, we use a sliding time window of size $\Delta \in \mathbb{N}$ to compute the changing STC or STC+ for each time window. 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 a long (possibly infinite) time range, the relationships, and thus, tie strengths, between participants change over time. Using the sliding time window approach, we can capture such changes.

In the following discussion, we first explain our streaming algorithm for the STC problem, and later in Sect. 4.3.4, we describe the necessary adaptions for the STC+ variant. Moreover, we assume 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 3 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 Fig. 3). 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.

4.3.1 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 $d(v) + d(w)$.*

Proof By adding $\{v, w\}$ in A, the number of new wedges in which a vertex v can be part is at most $d(v)$, and so we can at most create $d(v)$ new edges in $E(W)$. Analogously, for w, we can create at most $d(w)$ new edges. Therefore, v and w create at most $d(v) + d(w)$ edges in $E(W)$. Analogously, if we remove $\{v, w\}$, we destroy at most $d(v) + d(w)$ edges in $E(W)$.

The number of triangles that contain edge $\{v, w\}$ is bounded by $\min(d(v), d(w))$. If we add $\{v, w\}$ to A, we may close triangles of the form $\{u, v, w\}$. We remove one edge from $E(W)$ for each such triangle that is closed by edge $\{v, w\}$ in A; hence, at most $\min(d(v), d(w))$ edges are removed. On the other hand, by removing $\{v, w\}$ from $E(A)$, we can destroy at most $\min(d(v), d(w))$ triangles in A and have to insert corresponding edges in $E(W)$. □

4.3.2 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 2 shows our dynamic pricing approximation based on the non-dynamic 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 1 and following), which will be maintained and updated in subsequent calls of the algorithm. In the following, we show the general result that our algorithm gives a k-approximation of the MWVC after each update operation in a k-uniform hypergraph.

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 hypergraph (i.e. 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 2 calls for each request $r \in \sigma$ a corresponding procedure to update W and the current vertex cover C (line 32 and following). 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.*

In order to show Lemma 3, we first show the following properties for the Update procedure.

Lemma 4 *If the prices are fair, after calling* Update, *the prices are still fair. Furthermore, let C cover all hyperedges $E(W) \setminus F$. After* Update, *all hyperedges in F are covered by a vertex in C.*

Proof For each hyperedge $e \in F$ with a tight vertex, $p(e)$ is not changed, and e is covered by the tight vertex in C. Now, for each hyperedge $e \in F$ with no endpoint being tight, the price $p(e)$ is increased until one $u \in e$ is tight. Therefore, $s(u) \leq w(u)$ for all $u \in e$. Furthermore, at least one of the vertices $u \in e$ is added to C; hence, e is covered. □

Proof of Lemma 3 We prove the lemma for each procedure separately.

- InsEdge: Let e_n be the new hyperedge. If $u \in e_n$ is tight, the price of e_n will be $p(e_n) = 0$, and hence $s(u)$ for $u \in e_n$ does not change. Moreover, e_n is covered. If all $u \in e_n$ are not tight, the Update procedure will add $u \in e_n$ to C while maintaining fairness by Lemma 4.
- DelEdge: Let e_d be the hyperedge that is removed from $E(W)$. By updating $s(u)$ for $u \in e_d$ to their new values $s'(u) = s(u) - p(e_d)$ the vertices $u \in e_d$ might not be tight anymore. We remove them from C, which then might not be a valid vertex cover anymore. However, the prices are with $s'(u)$ fair. To repair the vertex cover C, we call Update with all edges incident to $u \in e_d$ whose other endpoints are not tight. These are precisely the hyperedges not covered by C. After running Update(F), C is a vertex cover, and fairness is maintained (Lemma 4).
- DecWeight: Decreasing the weight of $w(v)$ can affect the fairness. Hence, we have to adapt, i.e., lower, the prices of the hyperedges incident to v. By setting the prices of hyperedges e incident to v to $p(e) = 0$, fairness is restored. However, this might affect the tightness of neighbors of v because a formerly tight neighbor x connected by a hyperedge f might not be tight after setting $p(f) = 0$. Hence, we remove v and neighbors that are not tight anymore from C and collect all non-covered hyperedges from neighbors x of v into the set F. The set F contains all edges that are not covered. Calling Update on the set F leads to a valid vertex cover C while maintaining fairness (Lemma 4).
- IncWeight: Increasing the weight $w(v)$ does not affect fairness. However, it affects the tightness of v. Hence, if $v \in C$, we first remove v from C and add all hyperedges f incident to v for which all $x \in f$ are also not tight to the set F. All other edges not in F are covered by some $w \in C$. Again, by calling Update on the set F, we obtain a valid vertex cover C and maintain fairness (Lemma 4).

Theorem 3 *Algorithm* 2 *maintains a vertex cover with* $w(C) \leq k \cdot w(C^*)$, *where* C^* *is an optimal MWVC.*

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 C^* and fair prices, it holds that $\sum_{e \in E(W)} p(e) \leq w(C^*)$. To see this, consider the optimal MWVC C^* and

$$\sum_{u \in C^*} \sum_{e \in E(W) : u \in e} p(e) \leq \sum_{u \in C^*} w(u) = w(C^*).$$

Because C^* is a vertex cover, each edge contributes $p(e)$ (at least once) to the left hand side. Hence,

$$\sum_{e \in E(W)} p(e) \leq \sum_{u \in C^*} \sum_{e \in E(W) : u \in e} p(e) \leq \sum_{u \in C^*} w(u) = w(C^*).$$

Moreover, (2) for the vertex cover C and the computed prices, it holds that $\frac{1}{k} w(C) \leq \sum_{e \in E(W)} p(e)$. First note that $\sum_{e \in E : u \in e} p(e) = w(u)$ for all $u \in C$ because each node in C is tight. Therefore, we have

$$w(C) = \sum_{u \in C} w(u) = \sum_{u \in C} \sum_{e \in E(W) : u \in e} p(e) \leq k \cdot \sum_{e \in E(W)} p(e),$$

where the last inequality holds because for each k-uniform hyperedge e, we count $p(e)$ at most k times.

Consequently, the theorem follows from (1) and (2) as we have $w(C) \leq k \cdot \sum_{e \in E} p(e)$ and $k \cdot \sum_{e \in E} p(e) \leq k \cdot w(C^*)$. □

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 4 *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 2 Dynamic Pricing Approximation

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

Initialize and maintain vertex cover C and wedge graph W
Procedure UPDATE(*set of edges* F)**:**
  foreach $e \in F$ **do**
    if *there is* $u \in e$ *that is tight* **then** continue
    increase $p(e)$ until $u \in e$ is tight
  add newly tight vertices to C
Procedure DECWEIGHT(v, w_n)**:**
  $w(v) \leftarrow w_n$
  $C \leftarrow C \setminus \{v\}$
  $F' \leftarrow \delta(v)$ and initialize $F = \emptyset$
  foreach $e \in F'$ **do**
    $p(e) \leftarrow 0$
    if *all* $x \in e$ *are not tight* **then**
      $C \leftarrow C \setminus e$
      $F \leftarrow F \cup \{f \in E(G) \mid f \cap e \neq \emptyset$ and all $x \in f$ are not tight $\} \cup \{e\}$
  UPDATE(F)
Procedure DELEDGE($e_d \in E(W)$)**:**
  $E(W) \leftarrow E(W) \setminus \{e_d\}$
  update $s(u)$ for $u \in e_d$
  $C \leftarrow C \setminus e_d$
  $F \leftarrow \{f \in E(W) \mid e_d \cap f \neq \emptyset$ and all $x \in f$ are not tight$\}$
  UPDATE(F)
Procedure INSEDGE(e_n)**:**
  $E(W) \leftarrow E(W) \cup \{e_n\}$
  UPDATE($\{e_n\}$)
Procedure INCWEIGHT(v, w_n)**:**
  $w(v) \leftarrow w_n$
  if $v \in C$ **then**
    $C \leftarrow C \setminus \{v\}$
    $F \leftarrow \{e \in E(W) \mid v \in e$ and there is no tight vertex in $e\}$
    UPDATE(F)
foreach *update request* $r \in \sigma$ **do**
  Call the to r corresponding procedure
  $f \in$ {INSEDGE, DELEDGE, INCWEIGHT, DECWEIGHT}.

4.3.3 The streaming algorithm for TempMinSTC

Algorithm 3 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 2 to compute the TEMPMINSTC approximation for the current time window τ_t with $t \in [1, T(\mathcal{G}) - \Delta]$. Algorithm 3 outputs the weak edges based on the computed vertex cover C_t in line 11. It skips lines 7-11 if there are no changes in E_τ.

We now discuss the running time each iteration processing the current time window.

Theorem 5 *Let d_t^W (d_t^A) be the maximal degree in W (A, resp.) after iteration t of the while loop in Algorithm 3. The running time of iteration t is in $\mathcal{O}(\xi \cdot d_t^A \cdot (d_t^W)^2)$, with $\xi = \max\{|E_t^-|, |E_t^+|\}$.*

Proof We have the worst-case running time for a sequence σ_t that contains only DecWeight requests; see Theorem 2. Hence, the running time is in $\mathcal{O}(|\sigma_t| \cdot (d_t^W)^2)$. For the length of the sequence σ_t, we have the following considerations. By Lemma 2, we know that the number of InsEdge and DelEdge requests for one new or removed edge are in $\mathcal{O}(d_t^A)$, where d_t^A is the maximal degree in A. So any edge insertion into (or removal from) A during iteration t leads to at most $\mathcal{O}(d_t^A)$ requests. With $\xi = \max\{|E_t^-|, |E_t^+|\}$ it follows $|\sigma_t| \leq \xi \cdot d_t^A$. Hence, the result follows. □

It holds that $\xi = \max\{|E_t^-|, |E_t^+|\} \in \mathcal{O}(n^2)$, as at most all possible edges between the nodes of the aggregated graph can be inserted or removed. Furthermore, the maximum degree in the aggregated graph, d_t^A, is at most n. The maximum degree in the wedge graph, d_t^W, is upper-bounded by $2 \cdot d_t^A$. To see this, consider an edge $e = \{u, v\}$ in the aggregated graph A and the corresponding node n_e in the wedge graph W. For each edge incident to n_e, there must be one wedge in A containing e. The maximum number of such wedges is the sum of the degrees of u and v, which is at most twice the maximum degree. Consequently, the running time of iteration t is in $\mathcal{O}(n^2 \cdot (d_t^A)^3)$. However, as shown in Sect. 5, the aggregated graph is typically sparse, and the running time is highly efficient for real-world datasets.

Algorithm 3 Streaming algorithm for the STC/STC+ in temporal networks

Input: Stream of edges arriving in chronological order, $\Delta \in \mathbb{N}$
Output: Approximation of TempMinSTC/TempMinSTC+ for each time window of size Δ

```
1  Initialize t_s = 1, t_e = t_s + Δ − 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 τ_t = [t_s, t_e] such that ∀e ∈ E_τ it holds t(e) ∈ τ_t
5      Let E_τ^− (E_τ^+) be the edges removed from (inserted to) E_τ
6      if E_τ^− ≠ ∅ or E_τ^+ ≠ ∅ then
7          Use E_τ^− and E_τ^+ to update A and to obtain the update sequence σ_t
8          Call Algorithm 2 with σ_t to either update the
9            (i) wedge graph in the case of TempMinSTC, or
           (ii) wedge 3-uniform hypergraph in the case of TempMinSTC+
10         and obtain the MWVC approximation C_t
11         Output weak edges corresponding to nodes in the vertex cover C_t
12     Move time window forward by increasing t_s and t_e
```

Finally, the following results holds due to Theorem 3.

Theorem 6 *Algorithm* 3 *using a wedge hypergraph* (*variant* (*i*)) *maintains a* 2*-approximation of* TempMinSTC *for each time window of size* Δ.

4.3.4 The streaming algorithm for TempMinSTC+

The streaming algorithm presented in the previous section can be directly used for the TempMinSTC+ problem (variant (ii) in line 9). The main difference is that instead of a wedge graph, we need to maintain a wedge hypergraph H as described in Sect. 3.1.

When the time window moves forward, the updating the aggregated graphs is the same for the TempMinSTC and the TempMinSTC+ problem. After updating the aggregated graph, the wedge hypergraph H for solving the WeightedMinSTC+ problem needs to be updated, which is done similarly to the update of the wedge graph described in Sect. 4.3.1. Instead of normal edges, we insert and remove hyperedges defined by the wedges in the current aggregated graph A. For each new wedge $(v, \{u, w\}) \in \mathcal{W}(A)$, we add the hyperedge $\{n_{uv}, n_{vw}, n'_{uw}\}$ with n'_{uw} being a vertex corresponding to a potential new edge $\{u, w\} \in F \subseteq \binom{V}{2} \setminus E(A)$ in the aggregated graph. We call n'_{uw} a *new* vertex in $V(H)$. We add missing (new) vertices and remove isolated (new) vertices as we update the wedge hypergraph. Furthermore, the vertex weight function of the hypergraph is updated according to the edge weight changes in the aggregated graph.

When inserting or deleting edges from $E(A)$, or when changing edge weights in A, we can additionally consider the weight of the new vertices in the wedge hypergraph H. To this end, consider an edge $e = \{u, v\} \in E(A)$ that is modified, i.e., inserted, deleted, or its weight is changed due to the moving time window. Let W_e be the set of wedges in A that contain the edge e. For each wedge $(v, \{u, w\}) \in W_e$, there exists a corresponding hyperedge $\{n_{uv}, n_{vw}, n'_{uw}\} \in E(H)$ with n'_{uw} being a new node whose weight depends on the weight of e. Recall that the weight of n'_{uw} is defined as

$$w(n'_{uw}) = \alpha \cdot \sum_{(x,\{u,w\}) \in \mathcal{W}_f} \frac{w(\{u, x\}) + w(\{x, w\})}{|\mathcal{W}_f|},$$

where $\mathcal{W}_f = \{(x, \{u, w\}) \mid x \in V(A)\} \subseteq \mathcal{W}(A)$. When inserting or deleting e, the set $\mathcal{W}_f$ needs to be updated accordingly. Furthermore, when updating the weight $w(e) = w(\{u, x\})$ for $x = v$ (including after inserting or deleting e), the value of $w(n'_{uw})$ is updated as well.

As our dynamic MWVC algorithm in Sect. 4.3.2 already is stated for k-uniform hypergraphs it can be directly applied to update the MWVC in the wedge hypegraph leading to the following result.

Theorem 7 *Algorithm* 3 *using a wedge hypergraph* (*variant* (*ii*) *in line* 9) *maintains a* 3*-approximation of* TempMinSTC+ *for each time window of size* Δ.

5 Experiments

We compare the weighted and unweighted STC and STC+ on real-world temporal networks and evaluate the efficiency of our streaming algorithm. More specifically, we discuss the following research questions:

Q1. How do the weighted and non-weighted versions of the STC and STC+ compare to each other?

Q2. What is the impact of the parameter α on the weighted STC+?

Q3. How is the efficiency of our streaming algorithms?

5.1 Algorithms

We use the following algorithms for computing the weighted STC.

- `ExactW` and `ExactW+` are the weighted exact computation using the ILPs for the weighted STC and STC+ (see Sect. 3.2).
- `Pricing` and `Pricing+` use the non-dynamic pricing approximation in the wedge graph for the weighted STC and STC+.
- `DynAppr` and `DynAppr+` are our dynamic streaming algorithms for TempMinSTC and TempMinSTC+ based on Algorithm 3 for the time-window based computation of the weighted STC and STC+, respectively.
- `STCtime` and `STCtime+` are a baseline streaming algorithm for TempMinSTC and TempMinSTC+ that recompute the MWVC with the pricing method for each time window.

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

- `ExactNw` and `ExactNw+` are the exact computations using an ILP (see Adriaens et al. (2020); Veldt (2022)).
- `Matching` is the matching-based approximation of the unweighted vertex cover in the (non-weighted) wedge graph, see Sintos and Tsaparas (2014).
- `Matching+` is the adapted matching-based approximation of the unweighted vertex cover for the non-weighted wedge hypergraph.
- `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 Sintos and Tsaparas (2014).

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.

5.2 Data Sets

We use the following real-world temporal networks from different domains. The first three data sets are human contact networks from the *SocioPatterns* project. For these networks, the edges represent human contacts that are recorded using proximity sensors in twenty-second intervals. The contact networks are available at www.sociopatterns.org/.

- *Malawi* is a contact network of individuals living in a village in rural Malawi (Ozella et al. 2021). The network spans around 13 days.
- *Copresence* is a contact network representing spatial copresence in a workplace over 11 days (Génois and Barrat 2018).
- *Primary* is a contact network among primary school students over two days (Stehlé et al. 2011).

Furthermore, we use four online communication and social networks.

- *Enron* is an email network between employees of a company spanning over 3.6 years (Klimt and Yang 2004). The data set is available at www.networkrepository.com/.
- *Yahoo* is a communication network available at the *Network Repository* (Rossi and Ahmed 2015) (www.networkrepository.com/). The network spans around 28 days.
- *StackOverflow* is based on the stack exchange website StackOverflow (Paranjape et al. 2017). Edges represent answers to comments and questions. The network spans around 7.6 years. It is available at snap.stanford.edu/data/index.html.
- *Reddit* is based on the *Reddit* social network (Hessel et al. 2016; Liu et al. 2019). A temporal edge $(\{u, v\}, t)$ means that a user u commented on a post or comment of user v at time t. The network spans over 10.05 years. We used a subgraph from the data set provided at www.cs.cornell.edu/~arb/data/temporal-reddit-reply/.

When loading the data sets, we ignore possible self-loops at vertices. Table 2 shows the statistics of the data set. 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 Wu et al. (2016).

Table 2 Statistics of the data sets (*estimated)

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

5.3 Comparing weighted and non-weighted STC and STC+ (Q1)

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 3a 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 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 3b. `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 4 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.

Similarly to the comparison of the weighted and unweighted STC, we count the number of strong edges and the mean edge weight of strong edges of the first five data sets using the algorithms for the STC+. We set the weighting parameter

Table 3 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
Copres-ence	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

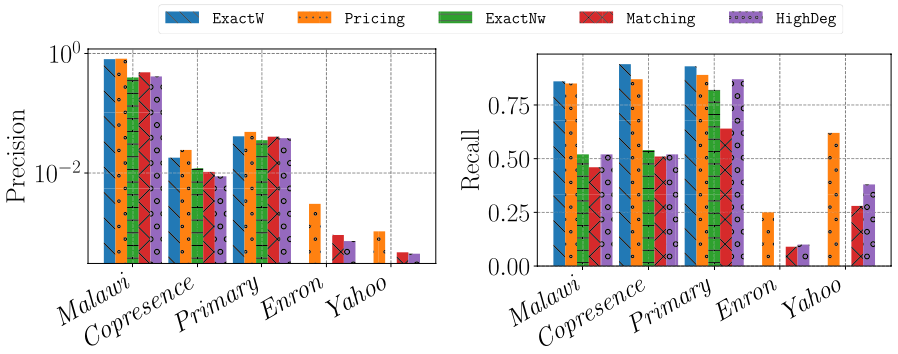


Fig. 4 Precision and recall for classifying the top-100 highest weighted edges in the aggregated graph as strong edges using the algorithms for the STC. The *y*-axis of precision is logarithmic

$\alpha = 0.5$ for newly inserted edges in the weighted version. Due to the increased number of variables, `ExactNw+` cannot finish the computation for *Primary*. Table 4a shows the percentage of strong edges of the number of edges in the input graph (i.e., not including newly inserted edges) computed using the different algorithms. For all data sets and algorithms, there are more strong edges compared to the standard STC variant, where the increase is strongest for *Copresence*. The reason is that by inserting additional weak edges, the number of strong edges can be increased. The unweighted `Matching+` approximation leads to slightly more strong edges compared to the weighted `Pricing+` algorithm because the latter tries to minimize the weight of the weak edges and the former the number of weak edges.

Moreover, we compare the quality of the STC+ by considering the mean weights of weak and strong edges shown in Table 4b. Compared to STC, the mean weights of the strong edges can be lower because more strong edges are included in the solution. However, for the *Copresence* network, the mean weights of the strong edges are higher for exact solutions. Similarly to the STC variant, the weighted STC+ approaches, `ExactW+` and `Pricing+`, lead to higher quality solutions with higher mean edge weights for the strong edges and lower mean edge weights for the weak edges.

5.4 Impact of parameter α on the weighted STC+ (Q2)

We computed the exact weighted STC+ using `ExactW+` for $\alpha \in \{0.001, 0.01, 0.1, 0.5, 0.75, 0.9\}$ and the *Malawi*, *Copresence*, and *Primary* data sets. Figure 5 shows the ratio of strong edges and the normalized mean edge weights of the strong edges. As expected, for lower values of α, more edges can be classified as strong because more wedges can be closed by adding weak edges. Due to the higher number of strong edges the mean edge weight decreases due to the inclusion of strong edges with low weight. For increasing α, the number of strong edges decreases, and the mean edge weight increases. Therefore, the α

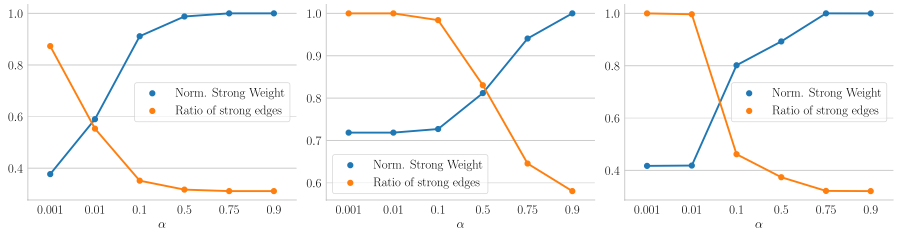
Table 4 Comparison of the weighted and non-weighted STC+ (OOT—out of time)

(a) Percentage of strong edges in aggregated graph.

Data set	Weighted		Non-weighted	
	`ExactW+`	`Pricing+`	`ExactNw+`	`Matching+`
Malawi	31.70	31.12	50.72	34.29
Copresence	83.04	38.00	90.73	57.27
Primary	37.39	26.46	OOT	32.25
Enron	OOT	3.66	OOT	5.57
Yahoo	OOT	12.35	OOT	14.03

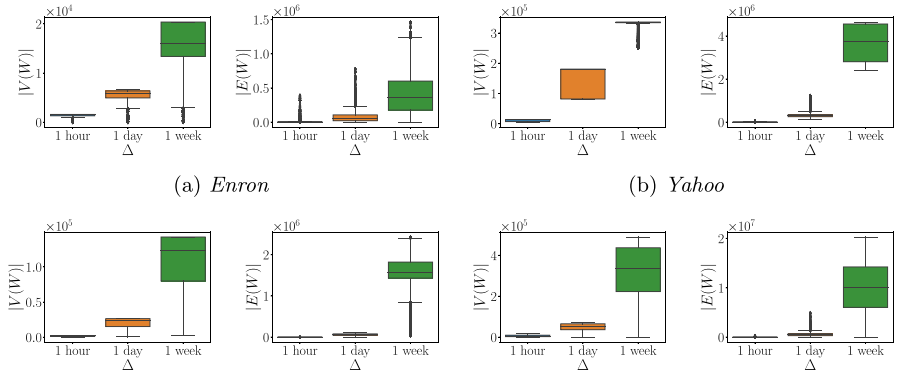
(b) Mean edge weights.

Data set	Weighted				Non-weighted			
	`ExactW+`		`Pricing+`		`ExactNw+`		`Matching+`	
	Weak	Strong	Weak	Strong	Weak	Strong	Weak	Strong
Malawi	21.33	883.97	18.61	905.97	242.02	343.19	198.56	479.16
Copresence	27.93	86.69	31.17	151.02	38.75	80.60	46.68	99.13
Primary	4.83	32.36	5.35	42.24	OOT	OOT	8.52	28.98
Enron	OOT	OOT	3.63	9.31	OOT	OOT	3.77	5.11
Yahoo	OOT	OOT	4.15	13.68	OOT	OOT	4.53	10.21



(a) *Malawi* (b) *Copresence* (c) *Primary*

Fig. 5 Effect of α on the ratio of strong edges and the normalizes sum of the weight of strong edges



(a) *Enron* (b) *Yahoo*

(c) *StackOverflow* (d) *Reddit*

Fig. 6 Boxplots for the sizes of the number of vertices $|V(W)|$ and edges $|E(W)|$ in the wedge graphs computed for the time windows of size Δ

parameter can be used to adjust the number of strong edges in a trade-off with the mean edge weight of strong edges.

5.5 Efficiency of the streaming algorithm (Q3)

In order to evaluate our streaming algorithms, we measured the running times on the *Enron, Yahoo, StackOverflow*, and *Reddit* data sets with time window sizes Δ of one hour, one day, and one week, respectively. Table 5 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 (see Table 5a). 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 Fig. 6a), and the dynamic algorithm can not make up for its additional complexity due

Table 5 Running times in seconds of the streaming alg. (OOT—out of time)

(a) Results for `DynAppr` and `STCtime`

	$\Delta = 1$ hour		$\Delta = 1$ day		$\Delta = 1$ week	
Data set	`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

(b) Results for `DynAppr+` and `STCtime+`

	$\Delta = 1$ hour		$\Delta = 1$ day		$\Delta = 1$ week	
Data set	`DynAppr+`	`STCtime+`	`DynAppr+`	`STCtime+`	`DynAppr+`	`STCtime+`
Enron	770.52	1 048.09	886.76	8 646.54	896.14	28831.41
Yahoo	36.69	11 026.78	168.21	OOT	260.21	OOT
StackOverflow	602.48	OOT	4 290.48	OOT	40 646.01	OOT
Reddit	5 622.78	OOT	OOT	OOT	OOT	OOT

to calling Algorithm 2. 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. Hence, the running times increase for both algorithms with increasing Δ. Figure 6 shows the sizes of the number of vertices $|V(W)|$ and edges $|E(W)|$ in the wedge graphs computed for the time windows of size Δ. The sizes increase with increasing Δ because more contacts happen in longer windows.

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.

Table 5b presents the results for `DynAppr+` and `STCtime+`. Similar to the previous case, `DynAppr+` demonstrates significantly faster performance than the baseline `STCtime+`, achieving orders of magnitude of speed-up. Even for the smallest dataset, *Enron*, our algorithm exhibits lower running time. It is worth noting that the number of edges remains the same as in the STC case, as one edge or hyperedge is introduced for each wedge in the aggregated graph of the current time window. However, the number of nodes in the wedge hypergraphs is typically much higher compared to the STC case, since many new nodes-up to one per hyperedge-are introduced. The additional nodes and their management generally result in higher running times compared to both `DynAppr` and `STCtime`.

6 Conclusion and future work

We generalized the STC and STC+ to weighted versions to include a priori knowledge in the form of edge weights representing empirical tie strength. We applied our new STC variants to temporal networks and showed that we obtained meaningful results. Our main contribution is our 2-approximation (3-approximation) streaming algorithm for the weighted STC (STC+, respectively) in temporal networks. We empirically validated its efficiency in our evaluation. Furthermore, we introduced a fully dynamic k-approximation of the MWVC problem in hypergraphs with k-uniform hyperedges that allows efficient updates as part of our streaming algorithm.

As an extension of this work, a discussion of further variants of the STC or STC+ can be interesting. For example, Sintos and Tsaparas (2014) introduced a variant with multiple relationship types. Efficient streaming algorithms for weighted versions of this variant are planned as future work.

Declarations

Conflict of interest The authors have no Conflict of interest to declare that are relevant to the content of this article.

References

Adriaens F, De Bie T, Gionis A et al (2020) Relaxing the strong triadic closure problem for edge strength inference. Data Min Knowl Discov 34:1–41

Ahmadian S, Haddadan S (2020) A theoretical analysis of graph evolution caused by triadic closure and algorithmic implications. In: 2020 IEEE International Conference on Big Data (Big Data), IEEE, pp 5–14

Bar-Yehuda R, Even S (1981) A linear-time approximation algorithm for the weighted vertex cover problem. J Algoritm 2(2):198–203

Bhattacharya S, Henzinger M, Italiano GF (2018) Deterministic fully dynamic data structures for vertex cover and matching. SIAM J Comput 47(3):859–887

Candia J, González MC, Wang P et al (2008) Uncovering individual and collective human dynamics from mobile phone records. J phys A: Math Theor 41:224015

Chen J, Molter H, Sorge M, et al (2018) Cluster editing in multi-layer and temporal graphs. In: 29th International Symposium on Algorithms and Computation, ISAAC, Schloss Dagstuhl–LZI, pp 24:1–24:13

Ciaperoni M, Galimberti E, Bonchi F et al (2020) Relevance of temporal cores for epidemic spread in temporal networks. Sci Rep 10(1):1–15

Easley DA, Kleinberg JM (2010) Networks, Crowds, and Markets - Reasoning About a Highly Connected World

Eckmann JP, Moses E, Sergi D (2004) Entropy of dialogues creates coherent structures in e-mail traffic. Proc Natl Acad Sci 101(40):14333–14337

Gallai T (1967) Transitiv orientierbare graphen. Acta Math Hung 18(1–2):25–66
Génois M, Barrat A (2018) Can co-location be used as a proxy for face-to-face contacts? EPJ Data Sci 7(1):11
Gilbert E, Karahalios K (2009) Predicting tie strength with social media. In: Proceedings of the 27th International Conference on Human Factors in Computing Systems, CHI, ACM, pp 211–220
Gilbert E, Karahalios K, Sandvig C (2008) 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
Gionis A, Oettershagen L, Sarpe I (2024) Mining temporal networks. Companion Proc ACM on Web Conf 2024:1260–1263
Granovetter MS (1973) The strength of weak ties. Am J Soc 78(6):1360–1380
Grüttemeier N, Komusiewicz C (2020) On the relation of strong triadic closure and cluster deletion. Algorithmica 82:853–880
Hanneke S, Xing EP (2006) Discrete temporal models of social networks. In: ICML Workshop on Statistical Network Analysis, Springer, pp 115–125
Hessel J, Tan C, Lee L (2016) Science, askscience, and badscience: On the coexistence of highly related communities. In: Proceedings of the International AAAI Conference on Web and Social Media, pp 171–180
Himmel A, Molter H, Niedermeier R et al (2017) Adapting the bron-kerbosch algorithm for enumerating maximal cliques in temporal graphs. Soc Netw Anal Min 7:1–16
Holme P, Saramäki J (2012) Temporal networks. Phys Rep 519(3):97–125
Holme P, Edling CR, Liljeros F (2004) Structure and time evolution of an internet dating community. Soc Net 26(2):155–174
Huang H, Tang J, Wu S et al (2014) Mining triadic closure patterns in social networks. 23rd International World Wide Web Conference. ACM, WWW, pp 499–504
Kahanda I, Neville J (2009) Using transactional information to predict link strength in online social networks. In: Proceedings of the International AAAI Conference on Web and Social Media, pp 74–81
Kempe D, Kleinberg JM, Kumar A (2002) Connectivity and inference problems for temporal networks. J Comput Syst Sci 64(4):820–842
Kleinberg J, Tardos E (2006) Algorithm design
Klimt B, Yang Y (2004) The Enron corpus: A new dataset for email classification research. In: European Conf. on Machine Learning, Springer, pp 217–226
Konstantinidis AL, Papadopoulos C (2020) Maximizing the strong triadic closure in split graphs and proper interval graphs. Discret Appl Math 285:79–95
Konstantinidis AL, Nikolopoulos SD, Papadopoulos C (2018) Strong triadic closure in cographs and graphs of low maximum degree. Theor Comput Sci 740:76–84
Kossinets G, Watts DJ (2006) Empirical analysis of an evolving social network. Science 311(5757):88–90
Latapy M, Viard T, Magnien C (2018) Stream graphs and link streams for the modeling of interactions over time. Soc Netw Anal Min 8:1–29
Lin N, Dayton PW, Greenwald P (1978) Analyzing the instrumental use of relations in the context of social structure. Sociol Method Res 7(2):149–166
Liu P, Benson AR, Charikar M (2019) Sampling methods for counting temporal motifs. In: Proceedings of the twelfth ACM international conference on web search and data mining, pp 294–302
Matakos A, Gionis A (2022) Strengthening ties towards a highly-connected world. Data Min Knowl Discov 36(1):448–476
Michail O (2016) An introduction to temporal graphs: an algorithmic perspective. Internet Math 12(4):239–280
Mislove A, Marcon M, Gummadi KP, et al (2007) Measurement and analysis of online social networks. In: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, pp 29–42
Moinet A, Starnini M, Pastor-Satorras R (2015) Burstiness and aging in social temporal networks. Phys Rev Lett 114(10):108701
Oettershagen L, Kriege NM, Morris C et al (2020) Classifying dissemination processes in temporal graphs. Big Data 8(5):363–378
Oettershagen L, Konstantinidis AL, Italiano GF (2022) Inferring tie strength in temporal networks. In: Machine Learning and Principles and Practice of Knowledge Discovery in Databases ECMLPKDD
Ozella L, Paolotti D, Lichand G et al (2021) Using wearable proximity sensors to characterize social contact patterns in a village of rural malawi. EPJ Data Sci 10(1):46

Paranjape A, Benson AR, Leskovec J (2017) Motifs in temporal networks. In: Proceedings of the tenth ACM international conference on web search and data mining, pp 601–610
Pham H, Shahabi C, Liu Y (2016) Inferring social strength from spatiotemporal data. ACM Trans Database Syst 41(1):7:1–7:47
Pyatkin A, Lykhovyd E, Butenko S (2019) The maximum number of induced open triangles in graphs of a given order. Optim Lett 13(8):1927–1935
Rossi RA, Ahmed NK (2015) The network data repository with interactive graph analytics and visualization. In: AAAI, https://networkrepository.com
Rozenshtein P, Gionis A (2019) Mining temporal networks. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp 3225–3226
Rozenshtein P, Tatti N, Gionis A (2017) 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, ACM, pp 1017–1025
Sintos S, Tsaparas P (2014) 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
Stehlé J, Voirin N, Barrat A et al (2011) High-resolution measurements of face-to-face contact patterns in a primary school. PloS One 6(8):e23176
Tantipathananandh C, Berger-Wolf TY (2011) Finding communities in dynamic social networks. 11th IEEE International Conference on Data Mining. IEEE Computer Society, ICDM, pp 1236–1241
Veldt N (2022) Correlation clustering via strong triadic closure labeling: Fast approximation algorithms and practical lower bounds. In: International Conference on Machine Learning, PMLR, pp 22,060–22,083
Viard T, Latapy M, Magnien C (2016) Computing maximal cliques in link streams. Theor Comput Sci 609:245–252
Wei HT, Hon WK, Horn P, et al (2018) 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), Schloss Dagstuhl–LZI, pp 27:1–27:14
Wu B, Yi K, Li Z (2016) Counting triangles in large graphs by random sampling. IEEE Trans Know Data Eng 28(8):2013–2026
Xiang R, Neville J, Rogati M (2010) Modeling relationship strength in online social networks. In: Proceedings of the 19th International Conference on World Wide Web, WWW, ACM, pp 981–990
Zhou L, Yang Y, Ren X, et al (2018) Dynamic network embedding by modeling triadic closure process. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, AAAI Press, pp 571–578

Authors and Affiliations

Lutz Oettershagen[1] · Athanasios L. Konstantinidis[2] · Giuseppe F. Italiano[3]

✉ Giuseppe F. Italiano
gitaliano@luiss.it

Lutz Oettershagen
lutz.oettershagen@liverpool.ac.uk

Athanasios L. Konstantinidis
a.konstantinidis@uoi.gr

[1] University of Liverpool, Liverpool, UK

[2] University of Ioannina, Ioannina, Greece

[3] Luiss University, Rome, Italy