Efficient Approximate Temporal Triangle Counting in Streaming with Predictions

Giorgio Venturin$^{\dagger 1}$, Ilie Sarpe$^{\dagger 2}$, and Fabio Vandin[1] ✉

[1] University of Padova, Padova, Italy
`giorgio.venturin@phd.unipd.it, fabio.vandin@unipd.it`
[2] KTH Royal Institute of Technology, Stockholm, Sweden
`ilsarpe@kth.se`

Abstract. Triangle counting is a fundamental and widely studied problem on static graphs, and recently on *temporal graphs*, where edges carry information on the timings of the associated events. *Streaming* processing and resource *efficiency* are crucial requirements for counting triangles in modern massive temporal graphs, with millions of nodes and up to billions of temporal edges. However, current exact and approximate algorithms are unable to handle large-scale temporal graphs. To fill such a gap, we introduce `STEP`, a scalable and efficient algorithm to approximate temporal triangle counts from a stream of temporal edges. `STEP` combines *predictions* to the number of triangles a temporal edge is involved in, with a simple sampling strategy, leading to scalability, efficiency, and accurate approximation of all eight temporal triangle types simultaneously. We analytically prove that, by using a sublinear amount of memory, `STEP` obtains unbiased and very accurate estimates. In fact, even noisy predictions can significantly reduce the variance of `STEP`'s estimates. Our extensive experiments on massive temporal graphs with up to billions of edges demonstrate that `STEP` outputs high-quality estimates and is more efficient than state-of-the-art methods.

Keywords: Temporal networks · Temporal triangle counting · Streaming algorithm.

1 Introduction

Temporal graphs model complex systems [13], including social networks [38] and databases [8], by associating each event in the system with its *timing* of occurrence. Temporal graph analysis provides a *deep understanding* of the underlying complex systems and their properties [14] through various problems such as temporal community detection [19], core decomposition [33], and pattern identification [10].

Temporal motifs and temporal triangles [21,29] are fundamental patterns defined by *i*) a subgraph representing a given *topological property*, *ii*) an ordering over the edges, capturing the timing of occurrence of the subgraph edges, and

[2] † denotes equal contribution.

iii) a temporal proximity constraint assuring that all events occur close in time. The *counts* of temporal motifs and temporal triangles are crucial for a plethora of graph analyses such as graph classification [39], anomaly detection [2], dense subgraph identification [36], and more [10]. Counting *temporal* motifs can be much more challenging than counting static subgraphs. In fact, identifying a single star-shaped temporal motif is **NP**-hard, unlike static graphs where all star-shaped subgraphs can be counted in polynomial time [20,37].

Given the importance of *temporal triangle counting*, both exact [22,30] and approximate [34] algorithms have been developed. However, exact approaches do not scale to modern-sized temporal graphs [29,9,22,18,30]. In addition, approximate *sampling* methods, which often need *full-access* to the input data, require the processing of large sample sizes due to their pessimistic worst-case analysis, which is extremely *inefficient* and impractical [41,35,20]. Overall, both exact and approximate temporal triangle counting methods require substantial resources to handle large temporal graphs, and designing scalable and efficient algorithms is an extremely challenging open problem.

Main contributions. We introduce `STEP`, a new algorithm for *approximate* temporal triangle counting in large temporal graphs. `STEP` processes temporal graph edges in a single pass *stream* [25], that is a challenging and practical setting. Streaming processing, in fact, enables analyzing massive temporal graphs, characterized by a high volume of interactions recorded over time [13,14], while limiting the memory available for storing the data. `STEP` uses a randomized sampling approach coupled with the information provided by a suitable *predictor* to identify and retain the most important edges over the stream. We prove that our design yields accurate estimates with sublinear memory and rigorous approximation guarantees, i.e., the output is a relative ε-approximation for small ε. Our extensive experimental evaluation shows that `STEP` saves up to 19$\times$ memory and 200$\times$ time compared to existing state-of-the-art methods while computing highly accurate estimates. Our key contributions are as follows:

1. We design `STEP`, a randomized, single-pass streaming algorithm to accurately estimate all temporal triangle counts simultaneously. `STEP` uses a sampling approach coupled with a predictor, resulting in low run time and memory usage.

2. We rigorously analyze `STEP`, proving that: *i*) it produces unbiased estimates independent of the prediction quality, *ii*) the predictor significantly decreases the variance of the estimates, and *iii*) estimates remain robust and of high quality even under reasonably noisy predictions.

3. We design a practical and efficient predictor that allows `STEP` to obtain high accuracy and small memory usage, despite being domain-agnostic.

4. We perform an extensive experimental evaluation on large temporal graphs to validate `STEP` and show that: *i*) it outperforms state-of-the-art (SotA) methods for temporal triangle counting, achieving highly accurate results while reducing resource usage by orders of magnitude; *ii*) `STEP` is the only method capable of obtaining accurate estimates on a three-billion-edge temporal graph; *iii*) `STEP` works in an online setting, using a predictor learned from historical data.

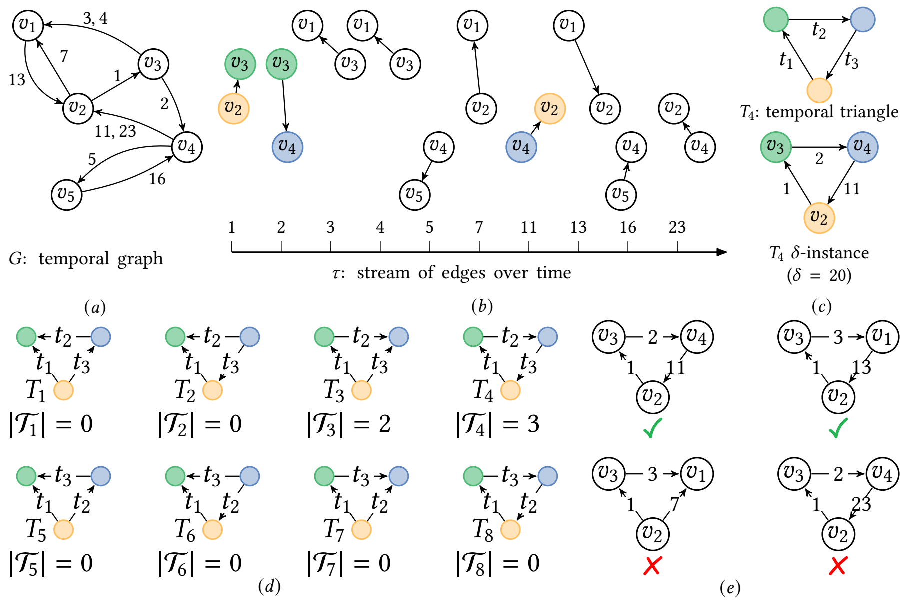

Fig. 1: (a): temporal graph $G = (V, E)$, with $V = \{v_1, \ldots, v_5\}$ and $E = \{(v_2, v_3, 1), \ldots\}$. (b): stream τ of the temporal edges of G. (c): the sequence $\langle(v_2, v_3), (v_3, v_4), (v_4, v_2)\rangle$ is a δ-instance of temporal triangle T_4 ($\delta = 20$). (d): all distinct temporal triangles and the number of their δ-instances in G ($\delta = 20$). Edge labels $t_i, i = 1, 2, 3$ (with $t_1 < t_2 < t_3$) denote the ordering of edges in the sequence σ (Def. 2). (e): some temporal triangles in G: δ-instances of T_4 for $\delta = 20$ are marked with ✓, while ✗ denotes edges that are not δ-instances.

2 Preliminaries

We start by introducing the key definitions and concepts used in our work.

Definition 1. *A* temporal graph *is a pair $G = (V, E)$ where V is a set of n vertices and $E = \{(u_1, v_1, t_1), \ldots, (u_m, v_m, t_m) : u_i, v_i \in V, u_i \neq v_i$ and $t_i \in \mathbb{R}^+\}$ is a set of m directed* temporal *edges. Each* temporal *edge $e = (u, v, t) \in E$ has a* timestamp *$t \in \mathbb{R}^+$ denoting the time of the (static) interaction (u, v).*

Fig. 1(a) shows an example of a temporal graph G with $n = 5$ and $m = 10$. We use the term *static edge* to denote an edge $(u, v) \in V^2$ with no timestamp. Similarly, a graph formed by static edges is a *static* graph.[3] Our focus is to count *temporal triangles* [29,20], fundamental patterns for analyzing temporal graphs.

Definition 2 ([29,20]). *A* temporal triangle *is a pair $T = ((V_T, E_T), \sigma)$ where (V_T, E_T) is a static graph with $|V_T| = 3$ vertices and $|E_T| = 3$ edges, and σ is an ordering of the edges in E_T.*

[3] The static graph of $G = (V, E)$ is obtained by considering all edges in E as static.

A temporal triangle T captures *both* structural (i.e., a triadic interaction) and temporal properties (i.e., the temporal ordering of edges). We denote each distinct temporal triangle with $T_i, i \in [8]$[4] (see Fig. 1(d)), while T will denote an arbitrary temporal triangle.

Definition 3. *Let $T = ((V_T, E_T), \sigma)$ be a temporal triangle where $\langle (x_1, y_1), (x_2, y_2), (x_3, y_3) \rangle$ is the sequence of edges of E_T ordered according to σ. Given a temporal graph $G = (V, E)$ and a time duration $\delta \in \mathbb{R}^+$ we say that a sequence of temporally-ordered edges $S = \langle (u_1, v_1, t_1), (u_2, v_2, t_2), (u_3, v_3, t_3) \rangle$ from E is a* δ-instance *of the temporal triangle T if: 1) there exists a bijection f on the vertices such that $f(u_i) = x_i, f(v_i) = y_i$ for $i \in \{1, 2, 3\}$; 2) the time-duration of the sequence S is at most δ, that is $t_3 - t_1 \le \delta$.*

Given a time-duration $\delta \in \mathbb{R}^+$, a δ-instance S represents an *occurrence* of the temporal triangle T within δ-time. See Fig. 1(c) and Fig. 1(e) for detailed examples. For $T_i, i \in [8]$, and a time-duration δ, we let $\mathcal{T}_i = \{\Delta \in E^3 : \Delta$ is a δ-instance of T_i in $G\}$ be the *set* of δ-instances of T_i in G, and $|\mathcal{T}_i|$ be the *count* of triangle T_i. Note that for a temporal graph with m temporal edges, the count $|\mathcal{T}_i|$ of a triangle T_i can be as large as $\Theta(m^3)$, and, in contrast to static graphs, m may not be polynomial in n (due to the edges timestamps).

Streaming model. We consider the following restrictive *streaming* computational model: *1)* the temporal graph G is accessed as a stream τ of temporal edges; *2)* edges on the stream τ are *temporally ordered*, i.e., if $e_1 = (u_1, v_1, t_1)$ precedes $e_2 = (u_2, v_2, t_2)$ on τ, then $t_1 < t_2$; and *3)* each temporal edge can be processed only *once*, in a 1-pass over τ. See Fig. 1(b) for an example of a stream.

The streaming model we consider is a challenging and restrictive computational model for processing temporal graphs, of high practical utility. Our computational model is much more restrictive compared with existing works, that allow for *multiple passes* over τ [41], or *complete random access* to the graph [30,29,9,18]. In fact, modern temporal graphs have massive sizes, e.g., they are collected from high-throughput systems such as IP networks or social networks, requiring a streaming access model, as in our model [13].

Computational problem. The *exact* computation of all temporal triangle counts is extremely challenging, inefficient, and resource demanding [30,22,29,9]. In contrast, we focus on computing *accurate estimates* of all counts $|\mathcal{T}_i|$ simultaneously, as formalized by the following problem.

Problem 1 (Temporal triangle estimation problem). Given a 1-pass stream τ of a temporal graph G, a time-duration $\delta \in \mathbb{R}^+$, an approximation error $\varepsilon > 0$, and a small constant η, output estimates c_i such that $\mathbb{P}[|c_i - |\mathcal{T}_i|| \ge \varepsilon |\mathcal{T}_i|] \le \eta, \forall i \in [8]$.

Prob. 1 requires the *simultaneous* computation of estimates c_i for triangle counts $|\mathcal{T}_i|, i \in [8]$, with guaranteed accuracy (i.e., relative ε-approximation) and bounded error probability η, over a challenging 1-pass stream τ. We also require that

[4] We use $i \in [a], a \in \mathbb{N}$ to denote $i \in \{1, \ldots, a\}$.

an algorithm for Prob. 1 must use *limited total memory* since temporal triangle counting is extremely memory-demanding [22,35,41]. Restricting the total memory to be sublinear is very common in streaming settings [42,27]. In our setting, we require a total memory *sublinear in* m_δ, i.e., the maximum number of temporal edges of the stream τ occurring in any time-window of length δ.

3 STEP algorithm

3.1 Overview

We first introduce the techniques and design choices behind our algorithm STEP. We design STEP to achieve sublinear memory guarantees (see Sec. 3.3), by building on ideas from state-of-the-art streaming algorithms for sublinear counting of *static* triangles and subgraphs [37]. That is, *1)* STEP stores, probabilistically, a *small sample* of edges from the stream τ; *2)* STEP computes unbiased estimates c_i *simultaneously* for each count $|\mathcal{T}_i|, i \in [8]$ based on the retained *random* sample. The above approach allows STEP to use sublinear memory and to obtain *unbiased* estimates c_i, i.e., $\mathbb{E}[c_i] = |\mathcal{T}_i|$. However, the estimates c_i may be far from $|\mathcal{T}_i|$, especially when the random sample retained by STEP is not representative. To compute estimates c_i close to their expectations $|\mathcal{T}_i|$, we build on ideas from the Algorithms with Predictions literature for *static graphs* [7,4]: *i)* we empower STEP with a predictor $\mathcal{Q}(\cdot)$ that enables the identification of important edges on the stream τ—yielding representative samples retained by STEP and highly accurate estimates c_i; *ii)* we design a predictor for the *simultaneous estimation* of all temporal triangle counts, relating STEP's accuracy with the quality of predictions of $\mathcal{Q}(\cdot)$. We prove that perfect predictions as well as noisy predictions result in very accurate estimates c_i and sublinear space complexity (Thm. 1), improving over the state-of-the-art (that does not leverage predictions). In addition, we also design a *practical* and efficiently computable predictor $\mathcal{Q}(\cdot)$ for STEP.

All missing proofs and subroutines are in our extended version [40].

3.2 Algorithm description

STEP leverages a randomized approach by sampling edges over the stream with a fixed probability $p \in (0, 1]$, similarly to state-of-the-art methods for estimating temporal subgraph counts [20,35,41]. In addition, STEP employs a *predictor* $\mathcal{Q}(\cdot)$, that classifies edges on the stream τ as either *heavy* or *light*. Heavy edges are those *predicted* to occur in many temporal triangles, and thus important to retain to collect a representative sample. The edge classification provided by $\mathcal{Q}(\cdot)$ is then used to obtain strong guarantees on small memory usage and high estimation accuracy. More in detail, STEP works as follows. First, it initializes two sets, H and S_L, for storing heavy and (sampled) light edges, respectively (line 1). It then initializes counters $c_{i,j}$ for each triangle type $T_i, i \in [8], j \in [0, 2]$ (line 2). All counters are used to output the unbiased estimates c_i for $|\mathcal{T}_i|$. STEP then processes the stream τ (line 3), and for each edge $e = (u, v, t)$: *1)* it removes

Algorithm 1: `STEP`

```
Input: Stream $\tau$ of temporal edges, time-duration $\delta$, predictor $\mathcal{Q}(\cdot)$, sampling
       probability $p \in (0,1]$.
Output: Estimates $c_i$ of $|\mathcal{T}_i|$ for $i \in [8]$.
1  $H \leftarrow \emptyset$; $S_L \leftarrow \emptyset$;
2  $c_{i,0} \leftarrow 0$; $c_{i,1} \leftarrow 0$; $c_{i,2} \leftarrow 0$ for $i \in [8]$;
3  foreach $e = (u,v,t) \in \tau$ do
4      $H \leftarrow$ CleanUp$(H, t-\delta)$; $S_L \leftarrow$ CleanUp$(S_L, t-\delta)$;
5      $\vee_{H,H}, \vee_{H,S_L}, \vee_{S_L,S_L} \leftarrow$ CollectWedges$(H, S_L, e)$;
6      $c_{i,0} \leftarrow$ UpdateCounts$(c_{i,0}, \vee_{S_L,S_L}, e)$ for $i \in [8]$;
7      $c_{i,1} \leftarrow$ UpdateCounts$(c_{i,1}, \vee_{H,S_L}, e)$ for $i \in [8]$;
8      $c_{i,2} \leftarrow$ UpdateCounts$(c_{i,2}, \vee_{H,H}, e)$ for $i \in [8]$;
9      if $\mathcal{Q}(e) = 1$ then $H \leftarrow H \cup \{e\}$;
10     else if BiasedCoin$(p) = true$ then $S_L \leftarrow S_L \cup \{e\}$;
11 return $c_i = \frac{c_{i,0}}{p^2} + \frac{c_{i,1}}{p} + c_{i,2}$ for $i \in [8]$;
```

edges from H and S_L with timestamps smaller than $t - \delta$ using the `CleanUp` procedure (line 4). The `CleanUp` procedure simply updates the sets H and S_L by retaining only edges $e' = (u', v', t')$ with t' within δ time from the timestamp of the current edge e, that is, $t - t' \leq \delta$. *2)* it collects all *wedges* in the sample $H \cup S_L$;[5] *3)* all collected wedges are partitioned into three subsets (line 5): $\vee_{H,H}$ (wedges with both edges in H), $\vee_{S_L,S_L}$ (both edges in S_L), and $\vee_{H,S_L}$ (one edge in H, the other in S_L); *3)* the counters $c_{i,j}$ are updated using the `UpdateCounts` procedure (lines 6-8), tracking occurrences of triangles T_i with 0, 1, or 2 heavy edges. `STEP` then calls the predictor $\mathcal{Q}(e)$ to determine whether e is heavy or light: if $\mathcal{Q}(e) = 1$, e is added to H (line 9); otherwise, e is added to S_L with probability p (line 10). Finally, `STEP` outputs estimates c_i for $i \in [8]$ by combining and weighting the counters $c_{i,j}, j = 0, 1, 2$ (line 11).

3.3 Analysis

Time complexity. First, we briefly consider the *expected* time complexity of `STEP`. Given the input to Algorithm 1, the expected time complexity of `STEP` is $\mathcal{O}(m(pm_\delta + |H|)^2)$, where: m_δ is the maximum number of edges over τ that occur in any time-duration of length δ, and $|H|$ is the number of heavy edges in H during the execution of the algorithm (see our extended version [40]). When $|H| = o(m_\delta)$ and $p \ll 1$ (e.g., as in our experiments), `STEP` is much more efficient than previous approaches (see Sec. 5). That is, `STEP` scales to large datasets, where previous approaches become impractical (see results in Sec. 4).
Unbiasedness. We prove that `STEP` computes *unbiased estimates* c_i of the counts $|\mathcal{T}_i|$, for $i \in [8]$, *independently* of the quality of the predictions of $\mathcal{Q}(\cdot)$.

[5] A wedge is a pair of distinct edges sharing a vertex.

Lemma 1. *Given a stream τ of a temporal graph $G = (V, E)$, a time-duration δ, and a predictor $\mathcal{Q}(\cdot)$, each estimate $c_i, i \in [8]$ reported by* `STEP` *is an unbiased estimate of the count $|\mathcal{T}_i|$, that is $\mathbb{E}[c_i] = |\mathcal{T}_i|$.*

Embedding predictions. We now analyze the impact of the predictor $\mathcal{Q}(\cdot)$ for our algorithm `STEP`. We first propose a practical model for a predictor, formalizing a *ranking predictor*. Our model is motivated by the fact that most machine learning models are highly optimized for ranking metrics, e.g., Kendall's tau or Spearman's correlation [43,11]. More in detail, a ranking predictor ranks edges $e \in \tau$ according to their importance for counts $|\mathcal{T}_i|$. We show analytically that both a *perfect* ranking predictor and a *noisy* one yield *accurate* estimates and *sublinear space complexity* for `STEP`.

Perfect predictor. Let $\omega(e, \mathcal{T}_i) = |\{\Delta : e \in \Delta, \Delta \in \mathcal{T}_i\}|$ be the number of triangles in $\mathcal{T}_i, i \in [8]$ containing edge $e \in E$, and $W(e) = \sum_{i \in [8]} \omega(e, \mathcal{T}_i)$ be the total edge weight of e. Let $\mathbf{W} \doteq \langle e_1^W, \ldots, e_m^W \rangle$ be the edges in E ordered by *non-increasing* values of their weights $W(e)$, ties broken arbitrarily. Given two distinct edges $e, e' \in E$, we use $e \prec e'$ to denote that e comes before e' in the ordering $\mathbf{W}$. With a slight abuse of notation, we denote with $e_{\prec_j}$ the edge in the j-th position in $\mathbf{W}$, and with $\mathbf{W}(e, \prec)$ the position of edge $e \in E$ in $\mathbf{W}$.

(Ranking predictor) Given an integer value $K > 0$, a *ranking predictor* $\mathcal{Q}(\cdot)_K$ is such that $\mathcal{Q}(e)_K = 1$ iff $\mathbf{W}(e, \prec) \leq K$.

A ranking predictor requires as unique input a parameter K, i.e., the number of edges to classify as heavy. Clearly, K corresponds to the maximum number of edges to be retained deterministically by `STEP`.[6] A ranking predictor does not require the knowledge of a threshold over edge weights $W(e), e \in E$ to classify heavy edges, in contrast with previous literature [7]. That is our predictor model is required to output the ranking $\mathbf{W}$ without having explicit access to the weights $W(e), e \in E$, as this would not be practical.

We next introduce a more practical *noisy* ranking predictor.

Noisy predictor. Given two parameters α and K, we let $\Pi(\{1, \ldots, m\})_{(\alpha,K)}$ be the set of permutations of m elements where three blocks of elements are fixed, i.e., blocks $[1, \ldots, K - \alpha - 1]$, $[K - \alpha, \ldots, K + \alpha]$ and $[K + \alpha + 1, \ldots, m]$. That is, elements from one block can only be permuted inside the same block.

(Noisy ranking predictor) Given a parameter $K > 0$ and $0 \leq \alpha \leq \min\{m - K + 1, K - 1\}$, an *$\alpha$-noisy K-ranking predictor* outputs $\mathbf{W}_\pi = \langle e_{\pi_1}^W, \ldots, e_{\pi_{|E|}}^W \rangle$, that is the vector $\mathbf{W}$ permuted according to $\pi \sim \mathfrak{U}(\Pi(\{1, \ldots, m\})_{(\alpha,K)})$, where $\mathfrak{U}(\cdot)$ denotes the uniform distribution over the elements of a set.

Therefore an α-noisy K-ranking predictor is such that it correctly classifies the top-$(K - \alpha - 1)$ edges in $\mathbf{W}$, and the edges with small weight $W(e)$ (i.e., all edges in position $j \geq K + \alpha + 1$). While, a noisy ranking predictor can be arbitrarily wrong in classifying edges in position $K - \alpha, \ldots, K + \alpha$ over $\mathbf{W}$. Where, α is a *noise parameter*: a larger value for α corresponds to a less accurate ranking predictor. Our noisy predictor definition closely reflects machine learning models

[6] The set of heavy edges H used by `STEP` has size trivially bounded by K.

or recommenders with high recall. Note that a 0-noisy predictor corresponds to a ranking predictor (without noise). Let $\nabla = \max_{i\in[1,m-1]}\{W(e_{\prec_i}) - W(e_{\prec_{i+1}})\}$ be the maximum difference of the weights for two adjacent edges in the vector $\mathbf{W}$. Finally let $\nabla_a = \nabla \cdot (a+1), a \geq 0$.

We now show that `STEP` computes a relative ε-approximation of all the temporal triangle counts $|\mathcal{T}_i|, i \in [8]$ with controlled error probability and sublinear memory usage in m_δ. We consider both perfect and noisy predictors.[7]

Theorem 1. *Consider an execution of* `STEP`*, with $\varepsilon > 0$ and $K = o(m_\delta)$. There exist constants $C > 1, \gamma \in (0, \frac{1}{2})$ such that:*

1. *when $\mathcal{Q}(\cdot)_K$ is a ranking predictor then if $p \geq \frac{C\sqrt{m_\delta}}{\varepsilon|\mathcal{T}_i|^{1/2-\gamma}}$,* `STEP` *uses $\mathcal{O}(\varepsilon^{-1}m_\delta^{3/2}/|\mathcal{T}_i|^{1/2-\gamma})$ memory in expectation;*
2. *when $\mathcal{Q}(\cdot)_K$ is an α-noisy K-ranking predictor for $\alpha \geq 1$ then if $p \geq \frac{C\sqrt{\nabla_\alpha m_\delta}}{\varepsilon|\mathcal{T}_i|^{1/2-\gamma}}$,* `STEP` *uses $\mathcal{O}\left(\varepsilon^{-1}m_\delta^{3/2}\sqrt{\nabla_\alpha}/|\mathcal{T}_i|^{1/2-\gamma}\right)$ memory in expectation.*

In both cases, `STEP` *is a one-pass streaming algorithm with $\mathbb{P}[\exists i \in [8] : |c_i - |\mathcal{T}_i|| \geq \varepsilon|\mathcal{T}_i|] \leq 1/3$ using sublinear memory.*

Consider case *1.* and the following event E = "count $|\mathcal{T}_i|^{1/2-\gamma}$ is sufficiently large, for each $i \in [8]$". Then Thm. 1 indicates that the sampling probability p can be set to a sufficiently small value, as $\sqrt{m_\delta}/|\mathcal{T}_i|^{1/2-\gamma} \ll 1$ under E. Consequently, the expected memory usage of `STEP` becomes sublinear in m_δ, aligning with established results in concentration theory [5]. Clearly, if E does not hold, then counts $|\mathcal{T}_i|$ are small enough to be obtained with high accuracy using the set H identified through $\mathcal{Q}(\cdot)$. Now, consider case *2.* from Thm. 1, under E, a noisy predictor increases `STEP`'s expected memory usage by a factor $\sqrt{\nabla_\alpha}$ (compared to case *1.*). That is `STEP` may miss some triangle counts due to noisy predictions, requiring a larger value for p to achieve the accuracy guarantees. Nonetheless, if the predictor effectively ranks important (heavy) edges (i.e., $\sqrt{\nabla_\alpha}$ is not too large), then `STEP` achieves accurate estimates with reduced variance compared to classical algorithms while maintaining an expected sublinear memory usage. In our extended version [40] we show, as a corollary of Thm. 1 case *1.*, that if the first K edges in $\mathbf{W}$ capture a sufficient number of triangles then $\mathrm{Var}[c_i]$ is a factor $\mathcal{O}(|\mathcal{T}_i|)$ smaller compared to when `STEP` does not use a predictor.

3.4 A simple and practical predictor

We now introduce a simple and practical noisy ranking predictor. That is, we describe a *temporal min-degree predictor* which can be built efficiently with a single pass on the input stream, and can be used within our algorithm `STEP`. We define the *temporal degree* of node $u \in V$ within a *time interval* $[t_a, t_b]$ as $d(u, t_a, t_b) = |\{(x, y, t') \in E : (x = u \text{ or } y = u) \text{ and } t' \in [t_a, t_b]\}|$. That is, the

[7] We assume that there exists an arbitrarily large constant R for which $|\mathcal{T}_i| = R \cdot |\mathcal{T}_j|, i, j \in [8]$. Such assumption can be avoided replacing $|\mathcal{T}_i|$ with $\sum_i |\mathcal{T}_i|$.

temporal degree is the number of edges incident to u within the given time interval. Next, given a temporal edge $e = (u, v, t) \in E$ and a time duration δ let $w_{\mathtt{m-d}}(e) = \min\{d(u, t-\delta, t+\delta), d(v, t-\delta, t+\delta)\}$ be the *temporal min-degree weight*. That is, $w_{\mathtt{m-d}}(e)$ is the minimum between the temporal degrees of the nodes of edge e. Intuitively, the temporal min-degree weight accounts for the temporal activity of nodes over the graph, which is crucial for our goal of designing a highly accurate predictor for `STEP`. Let $\mathbf{W}^{\mathtt{m-d}}$ be the edges $e \in E$ ordered by non-increasing values of their weight $w_{\mathtt{m-d}}(e)$, ties broken arbitrarily. Then, for any edge $e \in E$, the *temporal min-degree ranking predictor* classifies $\mathcal{Q}(e)_K = 1$ if e is within the first K edges of $\mathbf{W}^{\mathtt{m-d}}$, and $\mathcal{Q}(e)_K = 0$ otherwise. Therefore, the temporal min-degree predictor uses the temporal min-degree weights $w_{\mathtt{m-d}}(e)$ as a *proxy* for the unknown values $W(e)$ of a perfect predictor. Clearly, the temporal min-degree predictor may not be accurate when the rankings of $\mathbf{W}^{\mathtt{m-d}}$ and $\mathbf{W}$ do not align. Note that the temporal min-degree predictor can be computed extremely efficiently, with a single pass over the stream, and avoiding exact temporal triangle counting. Finally, note that our temporal min-degree predictor leverages both *structural* and *temporal* properties in the data—in contrast with predictors for static triangle counting [7,4], that do not consider time. In our extended version [40], we provide an empirical comparison for `STEP` coupled with different predictors, showing the superior performance of our temporal min-degree predictor compared with state-of-the-art static predictors.

4 Experimental evaluation

Our extensive experiments investigated the following questions:

Q1. How does `STEP` compare to SotA approaches in terms of accuracy of its estimates and computational resources (time and memory)?

Q2. What is the impact of the predictor $\mathcal{Q}(\cdot)$ on the estimates of `STEP`?

Q3. How does `STEP` perform in an *online setting*. Namely, when a predictor $\mathcal{Q}(\cdot)$ is learned on historical data, and then used on previously unseen data?

Datasets and environment. We considered four massive publicly available temporal graphs (Tab. 1), which are extremely challenging for Problem 1 and used by previous works [28,9,36,30]. More details on the used datasets and the setting of our experiments can be found in our extended version [40]. Our code is publicly available online.[8]

Baseline methods. We compared `STEP` with the following SotA algorithms: `Degeneracy` [30], an *exact* algorithm for computing the counts of all temporal triangles; `FAST-Tri` [9], an *exact* algorithm specifically tailored to temporal triangles; `MoTTo` [18], a recent SotA *exact* algorithm for counting 3-nodes 3-edges motifs, including triangles; and `EWS` [41], the SotA *approximate* method for solving Prob. 1.[9] We also compared with the sampling approach of `STEP` *without* a

[8] https://github.com/VandinLab/STEP.

[9] `EWS` requires two parameters p_{EWS} and q_{EWS} that we set as suggested by the authors [41] (see [40] for further details).

Table 1: Datasets. We report: $n = |V|$ the number of nodes; $m = |E|$ the number of temporal edges; the *precision* of the timestamps; and the total *timespan*.

Dataset	n	m	*precision*	*timespan*
Stackoverflow (SO)	$2.6\,M$	$63.5\,M$	sec	7.60 years
Bitcoin (BI)	$48.1\,M$	$113.1\,M$	sec	7.08 years
Reddit (RE)	$8.4\,M$	$636.3\,M$	sec	10.06 years
EquinixChicago (EC)	$11.1\,M$	$3.3\,B$	μ-sec	62.00 mins

Table 2: Peak RAM memory, in GB, of a representative run for the largest δ. *OOM* denotes out of memory.

Dataset	`NAIVE-S`	`STEP`$_{\texttt{P}}$	`STEP`$_{\texttt{TMD}}$	`EWS`	`Degeneracy`	`FAST-Tri`	`MoTTo`
SO	0.78	0.74	0.74	8.04	5.10	5.81	12.07
BI	1.70	1.81	1.74	27.05	15.40	17.43	34.08
RE	14.68	14.74	15.53	103.75	71.30	79.59	159.74
EC	63.46	62.70	63.47	*OOM*	*OOM*	*OOM*	*OOM*

predictor that we denote with `NAIVE-S`. Note that all exact approaches considered cannot process the input in streaming, requiring therefore large memory.
Memory and runtime. We measured the peak RAM memory (in GB) of each algorithm over a representative run. The runtime is an average over ten runs, unless otherwise stated. Since `EWS` counts triangles independently, its runtime is an average of the aggregated time to process all triangles, across ten runs.
Parameters. We select a small, a medium, and a large value for the parameter δ according to the precision of each dataset. We set on SO, BI and RE $\delta \in \{3\,600, 86\,400, 259\,200\}$, while for EC we set $\delta \in \{1 \times 10^5, 2 \times 10^5, 3 \times 10^5\}$. We set the parameter K to $\frac{m}{100}$. The sampling probability p_{NS} of `NAIVE-S` is set to obtain, in *expectation*, the same number of edges retained by `STEP` (i.e., $|S_L| + |H|$). All the parameters used in our experiments are in the extended version [40].
Ranking predictors. We considered two ranking predictors for `STEP`: *1)* a *perfect predictor* that exactly classifies the K edges with the highest weights $W(e), e \in E$ as defined in Sec. 3.3, and *2)* a *temporal min-degree* predictor, as described in Sec. 3.4. We denote the resulting methods with `STEP`$_{\texttt{P}}$ and `STEP`$_{\texttt{TMD}}$ respectively. The perfect predictor allows us to evaluate the performance of `STEP` when the edge weights correspond to the actual number of temporal triangles an edge is involved in—this is not of practical interest but provides a lower bound on the error of the estimates of `STEP`. The temporal min-degree predictor (`TMD`) is instead simple, general, domain-agnostic, and can be computed from simple structural properties in the data. The `TMD` uses the temporal min-degree as edge weight to classify the top-K most heavy edges to retain, as described in Sec. 3.4. The resulting method, `STEP`$_{\texttt{TMD}}$ is simple and of *practical* interest, given that the `TMD` can be computed efficiently.

4.1 Comparison with state-of-the-art methods

We first compared STEP with SotA baselines to answer question **Q1**. For such comparison, we fixed the parameters of all the approximation algorithms (STEP, NAIVE-S, and EWS) to ensure comparable runtime and MAE. Specifically, we ran STEP with the same sampling probability as EWS ($p = 0.01$) on all datasets except for SO, where $p = 0.1$ was used since STEP is much faster than EWS. When discussing results for STEP, we focus on $\texttt{STEP}_{\texttt{TMD}}$. Tab. 2 reports the peak memory usage for each algorithm, and Tab. 3 presents the runtime of all algorithms except $\texttt{STEP}_{\texttt{P}}$ and NAIVE-S (see [40] for additional results). Fig. 2 shows the accuracy of each approximate method on SO, BI and RE datasets for the largest values of δ (exact methods always report 0 MAE). While Fig. 3 reports the accuracy of the $\texttt{STEP}_{\texttt{P}}$, $\texttt{STEP}_{\texttt{TMD}}$ and NAIVE-S approaches on the EC dataset (EWS is not reported since it violates the RAM budget, see Tab. 2). Results for other values of δ are available in the extended version [40]. On the SO dataset, $\texttt{STEP}_{\texttt{TMD}}$ requires substantially less memory than EWS, Degeneracy, FAST-Tri and MoTTo while achieving more accurate estimates than EWS for all values of δ and for most temporal triangle counts. On the RE dataset, $\texttt{STEP}_{\texttt{TMD}}$ similarly demonstrates a significant reduction in memory usage compared to EWS, Degeneracy, FAST-Tri and MoTTo, and often provides higher-quality estimates than EWS. On the BI dataset, $\texttt{STEP}_{\texttt{TMD}}$ is much more memory efficient compared to EWS, Degeneracy, FAST-Tri and MoTTo. For larger values of δ, when the estimation problem becomes more challenging, $\texttt{STEP}_{\texttt{TMD}}$ obtains more accurate estimates than EWS. For smaller values of δ, the estimates provided by $\texttt{STEP}_{\texttt{TMD}}$ are comparable but slightly less accurate than those of EWS. In terms of runtime, $\texttt{STEP}_{\texttt{TMD}}$ consistently outperforms Degeneracy, FAST-Tri and MoTTo, and, in most cases, also EWS, with the exception of $\delta = 259\,200$ on the BI dataset. The BI dataset contains almost 40 billion δ-instances, most of which are counted deterministically by $\texttt{STEP}_{\texttt{TMD}}$ (over H). Hence, $\texttt{STEP}_{\texttt{TMD}}$ outputs tight estimates but requires a high execution time (see our extended version [40] for further analyses on the time-accuracy trade-off). Finally, on the EC dataset, that has more than 3 billion temporal edges, all SotA baselines cannot terminate their execution within the maximum memory allowance (200GB). Instead, $\texttt{STEP}_{\texttt{TMD}}$ requires less than 65GB of memory, achieves an average MAE below 0.1, and runs in less than ten minutes, even for the largest δ. In summary, these results show that $\texttt{STEP}_{\texttt{TMD}}$ enables the *efficient* and *accurate* estimation of all temporal triangle counts with remarkably *small memory usage*, especially on massive datasets where existing SotA approaches cannot scale their computation due to high resource usage.

4.2 Impact of the predictor

To answer **Q2**, we consider $\texttt{STEP}_{\texttt{P}}$, $\texttt{STEP}_{\texttt{TMD}}$, and NAIVE-S, and set their parameters so that they sample the same number of edges in expectation (see our extended version [40]). Fig. 2 and Fig. 3 show that both $\texttt{STEP}_{\texttt{P}}$ and $\texttt{STEP}_{\texttt{TMD}}$ provide much more accurate estimates than NAIVE-S on all datasets, with the

Table 3: Average runtime (in sec). "✗" denotes out of RAM memory (200 GB). Exact algorithms are run once due to their high runtime, hence we do not show their variance. The best runtime is in bold. `SU` denotes the speed-up of `STEP`$_{\texttt{TMD}}$ compared to each baseline (`N/A` is used when the speed-up cannot be computed).

Dataset	δ	`STEP`$_{\texttt{TMD}}$	`EWS`		`Degeneracy`		`FAST-Tri`		`MoTTo`	
		`Time`	`Time`	`SU`	`Time`	`SU`	`Time`	`SU`	`Time`	`SU`
SO	3600	**4.9 ± 0.0**	30.4 ± 0.8	6.2×	348.4	71.1×	15.1	3.1×	174.5	35.6×
	86400	**6.3 ± 0.1**	32.0 ± 0.5	5.1×	355.2	56.4×	41.8	6.6×	254.5	40.4×
	259200	**7.5 ± 0.1**	35.6 ± 0.9	4.7×	356.3	47.5×	76.3	10.2×	378.9	50.5×
BI	3600	**6.1 ± 0.1**	67.7 ± 5.1	11.1×	422.1	69.2×	189.0	31.0×	1113.7	182.6×
	86400	**43.8 ± 0.4**	115.9 ± 1.7	2.6×	421.3	9.6×	4287.7	97.9×	18045.1	412.0×
	259200	278.2 ± 5.5	**198.5 ± 5.9**	0.7×	424.8	1.5×	13804.3	49.6×	55494.2	199.5×
RE	3600	**69.7 ± 2.0**	570.7 ± 50.3	8.2×	18656.8	267.7×	1708.7	24.5×	6406.0	92.0×
	86400	**103.5 ± 4.4**	943.1 ± 67.6	9.1×	18528.1	179.0×	7479.5	72.3×	23085.0	223.0×
	259200	**164.9 ± 2.1**	1121.8 ± 79.1	6.8×	18950.0	115.0×	11165.8	67.7×	29680.1	180.0×
EC	1×10^5	**425.6 ± 16.8**	✗	`N/A`	✗	`N/A`	✗	`N/A`	✗	`N/A`
	2×10^5	**497.4 ± 8.6**	✗	`N/A`	✗	`N/A`	✗	`N/A`	✗	`N/A`
	3×10^5	**580.3 ± 0.9**	✗	`N/A`	✗	`N/A`	✗	`N/A`	✗	`N/A`

exception of $\delta = 3\,600$ for the BI dataset where `STEP`$_{\texttt{TMD}}$ and `NAIVE-S` are comparable (see our extended version [40]). Moreover, the variance of the estimates by both `STEP`$_{\texttt{P}}$ and `STEP`$_{\texttt{TMD}}$ is always smaller compared to `NAIVE-S`, especially for the larger datasets RE and EC. In terms of runtime, `NAIVE-S` is the fastest method (see [40]): in fact, `NAIVE-S` *counts fewer triangles* than `STEP`, yielding estimates with higher variance—highlighting a key trade-off, i.e., more accurate estimates require larger execution times for `STEP`. It is worth noting that on the EC dataset, `STEP`$_{\texttt{TMD}}$ requires more time to execute than `STEP`$_{\texttt{P}}$, in contrast to all other datasets. This is due to the structure of the EC dataset, on which the temporal min-degree does not provide a good proxy for the weights $W(e)$ of temporal edges (see our extended version [40]). Nevertheless, `STEP` still computes more accurate estimates than `NAIVE-S` while being highly memory efficient. To summarize, by employing a predictor `STEP`$_{\texttt{P}}$ and `STEP`$_{\texttt{TMD}}$ significantly improve the accuracy and reduce the variance of `NAIVE-S`'s estimates. However, such higher accuracy may lead to higher execution times compared to `NAIVE-S`, due to the processing of more occurrences.

4.3 Online estimation

To address **Q3**, we developed a practical approach for learning a predictor from a *training stream* of temporal edges (τ^{tr}) and used it to estimate temporal triangle counts on a *test stream* (τ^{ts}). The training stream τ^{tr} consists of the first 75% of edges appearing on the stream τ. The predictor is based on a threshold value ϕ, obtained from the temporal min-degree weight (see Sec. 3.4) of the K-th edge in the non-decreasing ordering induced by $w_{\texttt{m-d}}(e), e \in \tau^{tr}$. When processing the test stream, edges in τ^{ts} with temporal degree $w_{\texttt{m-d}}(e) \geq \phi$ are classified as

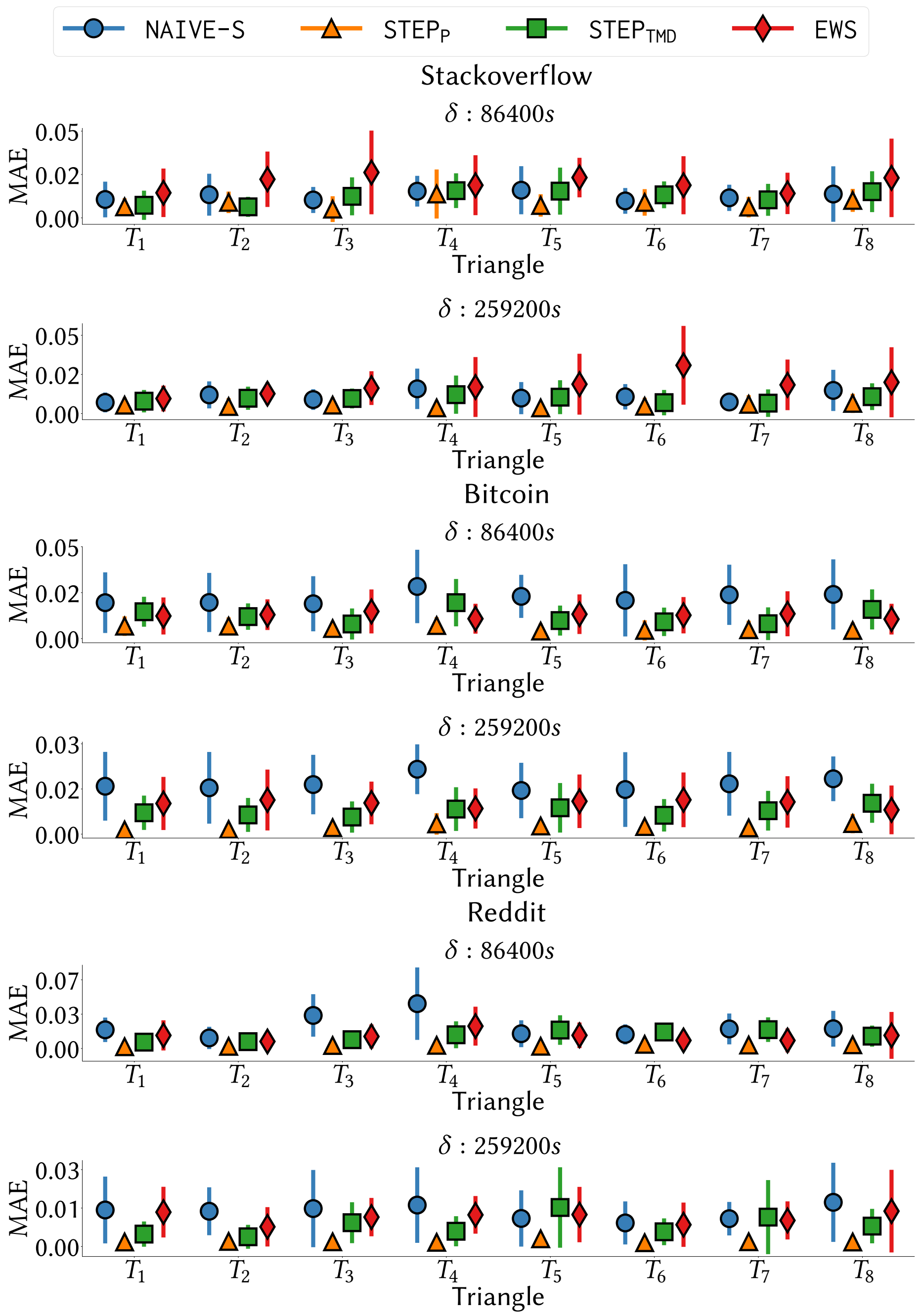

Fig. 2: MAE and standard deviation for `STEP`, `NAIVE-S` and `EWS` on SO, BI and RE datasets from Tab. 1, for the largest values of δ and for each temporal triangle (see Fig. 1(d)).

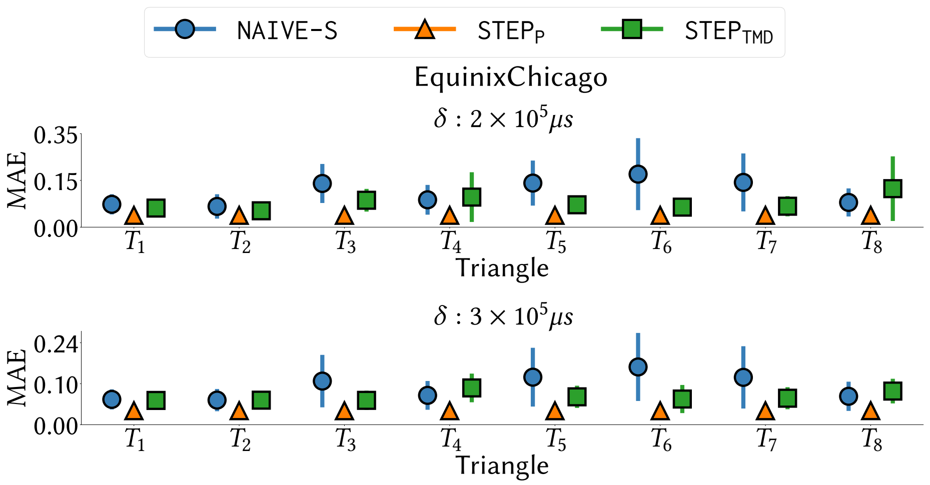

Fig. 3: MAE and standard deviation for STEP, NAIVE-S and EWS on EC dataset from Tab. 1, for the largest values of δ and for each temporal triangle (see Fig. 1(d)).

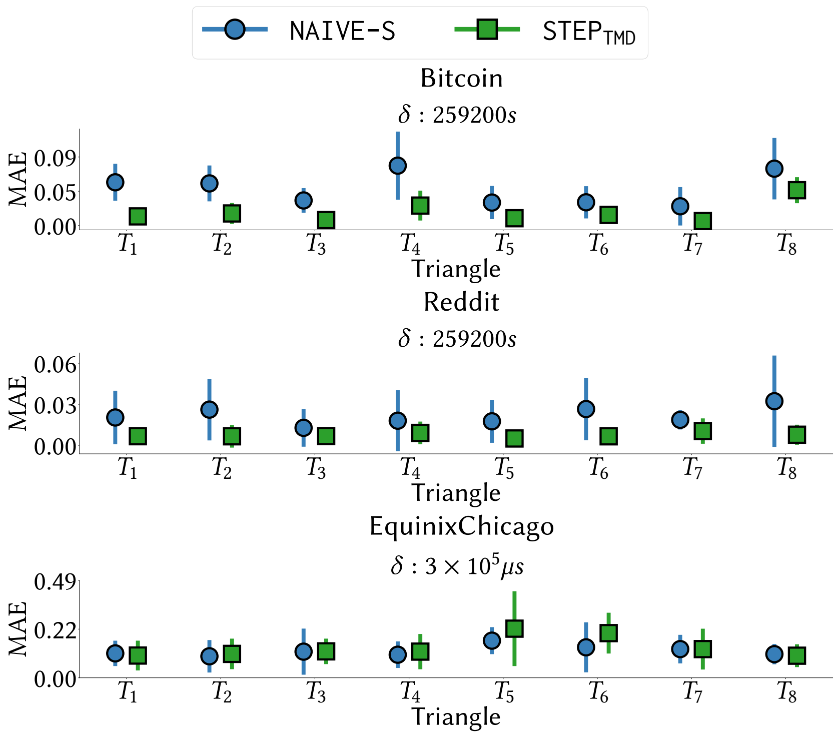

Fig. 4: Accuracy comparison (MAE and standard deviation) for online estimation over τ^{ts}, between NAIVE-S and $\texttt{STEP}_{\texttt{TMD}}$ (trained over the historical data τ^{tr}).

heavy and retained by the algorithm.[10] The underlying idea is that the threshold ϕ classifies important edges to retain over τ^{ts} whenever the training stream τ^{tr} is sufficiently representative (for τ^{ts}). The results for BI, RE and EC for the largest value of δ are shown in Fig. 4 (see the extended version [40]). We observe that $\texttt{STEP}_{\texttt{TMD}}$ provides more accurate estimates than `NAIVE-S` on the RE and BI datasets. On the EC dataset, $\texttt{STEP}_{\texttt{TMD}}$ and `NAIVE-S` achieve similar accuracy, as the learned predictor (i.e., ϕ) does not align well with a *perfect* classification over τ^{ts} (similarly to the results in Sec. 4.2). We study such aspects in the extended version [40]. Overall, our findings highlight that `STEP` effectively leverages simple predictors learned from historical data, often outperforming `NAIVE-S` in most configurations. Therefore, even under noisy and inaccurate predictions, `STEP` achieves good estimates, supporting its usage in practical applications.

5 Related work

To the best of our knowledge, our work is the first to solve the *temporal* triangle counting problem *using predictions*. We now survey the works most relevant to this paper; for overviews on temporal motifs and algorithms with predictions see [21,10,26].

Temporal motif and triangle counting. There exist several definitions of temporal motifs, including temporal triangles [21,23]. We adopt the definition introduced by Paranjape et al. [29], for which various counting algorithms exist. *Exact methods.* Paranjape et al. [29] introduced a method with complexity $\mathcal{O}(|E'|^{3/2} + m|E'|^{3/4})$, where $|E'|$ is the number of edges in the static graph of a temporal graph, that is impractical on large temporal graphs. Gao et al. [9] and Li et al. [18] improved the work in [29] introducing various pruning techniques yielding a time-complexity of $\mathcal{O}(mm_\delta^2)$, matching the complexity of the exhaustive enumeration algorithm in [22]. In fact, most recent works do not scale to large temporal graphs [18,9], as shown by our experimental evaluation (Sec. 4). Pashanasangi and Seshadhri [30] developed an exact method with complexity $\mathcal{O}(m\kappa \log m)$, where κ is the degeneracy of the static graph [24]: such an approach can be impractical on large real-world graphs where κ is in the order of hundreds or thousands [30]. Moreover, all existing exact approaches remain computationally impractical and extremely memory-intensive, as they require access to the temporal graph, and cannot work in streaming [9,30,18].
Approximate methods. Approximate approaches are based on randomized sampling. Most methods only approximate a *single* temporal motif count, either by collecting subgraphs within specific time windows [35,20] or by sampling temporal edges [41] or paths [28]. Some techniques can estimate *multiple* temporal motifs counts under specific constraints, such as shared static topology [34] or specific structure [32], but cannot process the graph as a stream. To our best knowledge, `EWS` by Wang et al. [41] is the only approximation algorithm designed for streaming processing. `EWS` uses edge sampling to obtain an estimate for an

[10] The predictor evaluates if $e = (u, v, t)$ should be retained or not at time $t + \delta$.

individual triangle count. However, EWS requires large computational resources, such as memory, not scaling on large temporal graphs (see Sec. 4).

Algorithms with Predictions (AwP) in static graphs. The AwP framework introduced in [26] enhances classical combinatorial algorithms with predictions obtained, for example, from machine learning models trained on historical data. Classical combinatorial algorithms benefit from predictions by improving their efficiency, e.g., runtime or memory usage, and retaining their worst-case complexity. Such new framework has been applied to clustering [15], graph problems [17], and more [1,6,3]. Related to our work, Chen et al. [7] developed a predictor-based sampling algorithm to estimate triangle and four-cycle counts in *static* graph streams. In addition to target *static* graphs, the work of Chen et al. [7] relies on the impractical assumption of a predictor knowing the number of triangles an edge participates in. Clearly, such an assumption is impractical: *i*) a perfect predictor can be obtained only by solving the triangle counting problem exactly; and *ii*) it cannot model the complex predictors used in practice. Recently, Boldrin and Vandin [4] improved the work in [7] and proposed a simple, domain-independent predictor that can be obtained with a single pass over the stream. Unfortunately, the idea in [4] is not suitable for solving the *temporal* triangle counting problem, as it makes STEP very inefficient, especially compared with our novel predictor (see our extended version [40]).

6 Conclusion

We studied the problem of counting temporal triangles in a stream of temporal edges. We introduced STEP, a sampling algorithm enhanced with a predictor, which provides highly accurate estimates while using minimal computational resources compared to SotA approaches. To the best of our knowledge, STEP is the *first algorithm* for temporal triangle counting using predictions. Experimental results show that STEP is much faster than SotA exact methods and requires significantly less resources than approximate SotA streaming algorithms, often obtaining more accurate estimates. Finally, we show how to efficiently compute a simple predictor, that can be also used for an online processing of the graph.

Future research includes the development of more advanced and domain-dependent predictors (e.g., learning specific edge-weights for the classification of important edges to retain over the stream), and extending STEP's approach to other temporal motifs [32,34].

Acknowledgments. This research is funded by the Ministry of University and Research within the Complementary National Plan PNC-I.1 "Research initiatives for innovative technologies and pathways in the health and welfare sector, D.D. 931 of 06/06/2022, PNC0000002 DARE - Digital Lifelong Prevention CUP: B53C22006440001" and PRIN Project n. 2022TS4Y3N "EXPAND: scalable algorithms for EXPloratory Analyses of heterogeneous and dynamic Networked Data". This research is also funded by the ERC Advanced Grant REBOUND (834862), and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

6.1 Disclosure of Interests.

The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Azar, Y., Panigrahi, D., Touitou, N. Online graph algorithms with predictions. SODA 2022.
2. Belth, C., Zheng, X., Koutra, D. Mining persistent activity in continually evolving networks. KDD 2020.
3. Bernardini, G., Lindermayr, A., Marchetti-Spaccamela, A., Megow, N., Stougie, L., Sweering, M. A universal error measure for input predictions applied to online graph problems. NeurIPS 2022.
4. Boldrin, C., Vandin, F. Fast and accurate triangle counting in graph streams using predictions. ICDM 2024.
5. Boucheron, S., Lugosi, G., Bousquet, O. Concentration Inequalities. Springer 2004.
6. Chen, J., Silwal, S., Vakilian, A., Zhang, F. Faster fundamental graph algorithms via learned predictions. ICML 2022.
7. Chen, J.Y., Eden, T., Indyk, P., Lin, H., Narayanan, S., Rubinfeld, R., Silwal, S., Wagner, T., Woodruff, D.P., Zhang, M. Triangle and four cycle counting with predictions in graph streams. ICLR 2022.
8. Debrouvier, A., Parodi, E., Perazzo, M., Soliani, V., Vaisman, A. A model and query language for temporal graph databases. VLDB Journal 2021.
9. Gao, Z., Cheng, C., Yu, Y., Cao, L., Huang, C., Dong, J. Scalable motif counting for large-scale temporal graphs. ICDE 2022.
10. Gionis, A., Oettershagen, L., Sarpe, I. Mining temporal networks. WWW 2024.
11. Heeg, F., Scholtes, I. Using causality-aware graph neural networks to predict temporal centralities in dynamic graphs. NeurIPS 2024.
12. Hessel, J., Tan, C., Lee, L. Science, AskScience, and BadScience: On the Coexistence of Highly Related Communities. ICWSM 2016.
13. Holme, P., Saramäki, J. Temporal networks. Phys. Rep. 2012.
14. Holme, P., Saramäki, J. Temporal Network Theory. Springer Int. Publishing 2023.
15. Jiang, S.H.C., Liu, E., Lyu, Y., Tang, Z.G., Zhang, Y. Online facility location with predictions. ICLR 2022.
16. Kondor, D., Csabai, I., Szüle, J., Pósfai, M., Vattay, G. Inferring the interplay between network structure and market effects in bitcoin. New J. of Physics 2014
17. Lattanzi, S., Ola, S., Sergei, V. Speeding up Bellman-Ford via minimum violation permutations. ICML 2023.
18. Li, J., Qi, J., Huang, Y., Cao, L., Yu, Y., Dong, J. Motto: Scalable motif counting with time-aware topology constraint for large-scale temporal graphs. CIKM 2024.
19. Lin, L., Yuan, P., Li, R.H., Zhu, C., Qin, H., Jin, H., Jia, T. Qtcs: Efficient query-centered temporal community search. PVLDB 2024.
20. Liu, P., Benson, A.R., Charikar, M. Sampling methods for counting temporal motifs. WSDM 2019.
21. Liu, P., Guarrasi, V., Sariyuce, A.E. Temporal network motifs: Models, limitations, evaluation. TKDE 2021.
22. Mackey, P., Porterfield, K., Fitzhenry, E., Choudhury, S., Chin, G. A chronological edge-driven approach to temporal subgraph isomorphism. Big Data 2018.

23. Mang, Q., Chen, J., Zhou, H., Gao, Y., Zhou, Y., Peng, R., Fang, Y., Ma, C. Efficient historical butterfly counting in large temporal bipartite networks via graph structure-aware index. arXiv 2024.
24. Matula, D.W., Beck, L.L. Smallest-last ordering and clustering and graph coloring algorithms. JACM 1983.
25. McGregor, A. Graph stream algorithms: a survey. SIGMOD 2014.
26. Mitzenmacher, M., Vassilvitskii, S. Algorithms with predictions. CACM 2022.
27. Muthukrishnan, S., et al. Data streams: Algorithms and applications. In: Foundations and Trends®. Theor. Comput. Sci. 2005.
28. Pan, Y., Bhalerao, O., Seshadhri, C., Talati, N. Accurate and fast estimation of temporal motifs using path sampling. ICDM 2024.
29. Paranjape, A., Benson, A.R., Leskovec, J. Motifs in temporal networks. WSDM 2017.
30. Pashanasangi, N., Seshadhri, C. Faster and generalized temporal triangle counting, via degeneracy ordering. KDD 2021.
31. Porter, A., Mirzasoleiman, B., Leskovec, J. Analytical models for motifs in temporal networks. WWW 2022.
32. Pu, J., Wang, Y., Li, Y., Zhou, X. Sampling algorithms for butterfly counting on temporal bipartite graphs. arXiv 2023.
33. Qin, H., Li, R.H., Yuan, Y., Wang, G., Qin, L., Zhang, Z. Mining bursting core in large temporal graphs. PVLDB 2022.
34. Sarpe, I., Vandin, F. Oden: simultaneous approximation of multiple motif counts in large temporal networks. CIKM 2021.
35. Sarpe, I., Vandin, F. Presto: Simple and scalable sampling techniques for the rigorous approximation of temporal motif counts. SDM 2021.
36. Sarpe, I., Vandin, F., Gionis, A. Scalable temporal motif densest subnetwork discovery. KDD 2024.
37. Seshadhri, C., Tirthapura, S. Scalable subgraph counting: The methods behind the madness. WWW 2019.
38. Tang, J., Musolesi, M., Mascolo, C., Latora, V. Temporal distance metrics for social network analysis. SIGCOMM 2009.
39. Tu, K., Li, J., Towsley, D., Braines, D., Turner, L.D. gl2vec: learning feature representation using graphlets for directed networks. ASONAM 2019.
40. Venturin, G., Sarpe, I., Vandin, F. Efficient Approximate Temporal Triangle Counting in Streaming with Predictions (extended version). arXiv 2025.
41. Wang, J., Wang, Y., Jiang, W., Li, Y., Tan, K.L. Efficient sampling algorithms for approximate motif counting in temporal graph streams. arXiv 2022.
42. Wang, P., Qi, Y., Sun, Y., Zhang, X., Tao, J., Guan, X. Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage. PVLDB 2017.
43. Zhang, T., Fang, J., Yang, Z., Cao, B., Fan, J. Tatkc: A temporal graph neural network for fast approximate temporal katz centrality ranking. WWW 2024.