Fast and Accurate Triangle Counting in Graph Streams Using Predictions

Anonymous authors

***Abstract*—In this work, we present the first efficient and practical algorithm for estimating the number of triangles in a graph stream using *predictions*. Our algorithm combines waiting room sampling and reservoir sampling with a *predictor* for the *heaviness* of edges, that is, the number of triangles in which an edge is involved. As a result, our algorithm is fast, provides guarantees on the amount of memory used, and exploits the additional information provided by the predictor to produce highly accurate estimates. We also propose a simple and domain-independent predictor, based on the degree of nodes, that can be easily computed with one pass on a stream of edges when the stream is available beforehand.**

Our analytical results show that, when the predictor provides useful information on the heaviness of edges, it leads to estimates with reduced variance compared to the state-of-the-art, even when the predictions are far from perfect. Our experimental results show that, when analyzing a single graph stream, our algorithm is faster than the state-of-the-art for a given memory budget, while providing significantly more accurate estimates. Even more interestingly, when sequences of hundreds of graph streams are analyzed, our algorithm significantly outperforms the state-of-the-art using our simple degree-based predictor built by analyzing only the first graph of the sequence.

***Index Terms*—Triangle Counting, Sampling, Algorithms with Predictions, Graph Stream Mining**

I. Introduction

Counting the number of triangles is a fundamental primitive in the analysis of graphs, with applications ranging from community detection [3] and anomaly detection [2], [18] to molecular biology [20]. In most applications the exact computation of the number of triangles is unfeasible, due to the massive size of the data. For this reason, one has often to resort to efficient algorithms that provide high-quality approximations, that can be used in place of exact values in subsequent analyses. One of the main requirements for such algorithms is that they use a limited amount of memory, for example storing only a fraction of the edges of the graph to not exceed a given memory budget.

In some applications the graphs of interest are not only massive, but they are also observed as a stream of edges that occur in arbitrary order. In such situations one is interested in keeping *high-quality* approximations *at every time* during the stream. Even when the data is not observed as a stream, analyzing a graph through one (or few) passes on its edges leads to efficient algorithms.

A lot of work has been done on approximate triangle counting in streams, often using sampling (see Section II). For example, De Stefani et al. [32] introduced the use of *reservoir sampling* to keep the memory bounded *with certainty* and, at the same time, make better use of the available memory. More recently Shin et al. [29] introduced the notion of *waiting room sampling*, that allows to take advantage of temporal localities of triangles observed in most applications by keeping the most recent edges in the stream in a waiting room. While such algorithms provide accurate and efficient solutions for counting triangles in data streams, there is still room for improvement. For example, such algorithms cannot adapt to the data distribution that is specific to each application, and that may be captured by using a *predictor*, such as a machine learning model, for example, providing additional and relevant information on the graph [22].

Contributions. We introduce `Tonic` (*T*riangles c*O*u*N*ting with pred*IC*tions), a suite of novel efficient algorithms for fast and accurate triangle counting in graph streams. `Tonic` can be instantiated for both insertion-only and fully dynamic streams with both edge insertions and deletions. For ease of presentation and due to space constraints we focus on the insertion-only variant of `Tonic`; the variant of `Tonic` for fully dynamic streams is described in the extended version of this paper [**?**]. Our algorithms use uniform sampling schemes to store a small sample of the stream while making sure to not exceed the allowed memory budget.

- `Tonic` is the first *fast* and *accurate* algorithm for approximating the number of triangles in graph streams using *predictions*. `Tonic` combines such predictions with reservoir sampling and waiting room sampling to provide high-quality estimates of both the *global* number of triangles and the *local* number of triangles incident to each vertex. `Tonic` can use any predictor providing some measure of *heaviness* for edges, defined as the number of triangles an edge is involved in. By using a predictor, `Tonic` focuses on the most relevant edges for triangle counting, and it adapts to the data distribution specific to the application. Our analysis shows that the predictor leads to improved estimates when it provides useful information on heavy edges, even if its predictions are far from perfect.
- We propose a very simple and application independent predictor, based on the degree of nodes, to be used in `Tonic`. Such predictor can be learned with one pass of the stream and can be easily stored. This is in contrast with previous work exploring the use of *heaviness* predictors for triangle counting in graphs streams [6], and makes `Tonic` the first *practical* approach for triangle counting in data streams with predictions.
- We conduct an extensive experimental evaluation on very large graph streams and on sequences of graph streams.

The results show that, when analyzing a single graph stream, `Tonic` is faster than the state-of-the-art for a given memory budget, while providing significantly more accurate estimates. The improvement is even more significant for sequences of hundreds of graph streams, where `Tonic` substantially outperforms the state-of-the-art using our simple degree-based predictor built analyzing only the *first* graph stream.

II. Related Works

The problem of counting global and local triangles in a graph has been extensively studied in the last decades [11], [19], [35], [36], in many different settings [23]–[26], [34]. Due to space constraints, we discuss here the works mostly related to ours, focusing on sampling approaches to estimate triangle counts in graph streams, and refer to the surveys [1], [15] for a more in depth presentation of other approaches. Sampling approaches fall into two main categories: *fixed memory*, where edges are sampled without exceeding a given memory budget, and *fixed probability*, where edges are sampled with a given fixed probability. While the two approaches are related (by appropriately setting the sampling probability in fixed probability approaches), the latter does not guarantee that the memory budget is not exceeded. Since our focus is on a fixed memory budget, we discuss only such kind of approaches.

Buriol et al. [5] presented a suite of sampling algorithms, resulting in $(1+\epsilon)$-approximation of the global triangles. Pavan et al. [27] introduced a one-pass stream efficient approach that runs different estimators to approximate the count of triangles, providing guarantees on the memory used. Jha et al. [10] provided an algorithm for approximating the global number of triangles with a single pass through a graph stream using the birthday paradox. De Stefani et al. [32] proposed *Trièst*, a suite of algorithms for counting the global and local number of triangles in fully-dynamic streams using reservoir sampling (see Section III-C) to keep edges within a fixed memory budget, and random pairing [7] to maintain a bounded-size uniform random sample of the edges in a fully dynamic stream. Shin et al. [29] and Lee et al. [16] introduced the idea of waiting room sampling (see Section III-A), that consists in storing the most recent edges, based on the empirical observation that triangles have edges spanning a short interval in the stream. Shin et al. [30], [31] presented *ThinkD*, a suite of algorithms for handling fully dynamic graph streams with random pairing, that builds on the scheme introduced by [32] resulting in highly accurate estimates.

The use of predictions about the input has been recently formalized in the *algorithms with predictions* framework (see the survey by Mitzenmacher and Vassilvitskii [22]), and several algorithms with predictions for well-studied problems have been proposed [9], [12], [21]. The problem of counting *global* triangles in a *insertion-only* graph stream has been recently studied in such framework by Chen et al. [6]. They proposed one-pass streaming algorithms for estimating the number of global triangles and four cycles using an oracle that provides predictions about the heaviness of edges. In this regard, the algorithm from [6] is similar to our algorithm for insertion-only graph streams, but there are three crucial differences. First, we also propose a variant of our algorithm for fully dynamic graph streams, while [6] limit to insertion-only graph streams. Second, the algorithm they propose and analyze uses a fixed probability approach, with no guarantees to respect the allowed memory budget and requiring knowledge of the number of edges in the stream to compute the sampling probability. Third, they assume that the oracle is available to the algorithm, but do not propose practical and efficiently computable oracles, that is instead one of our main contributions. As a result, our algorithm `Tonic` is the first efficient and practical approach that uses predictions for estimating the number of triangles in (fully dynamic) graph streams, as shown by our experimental evaluation (see Section V).

III. Preliminaries

We now introduce the main notions used in this paper. We consider an undirected graph $G = (V, E)$ with no self-loops and no multiple edges, where V and E are the set of nodes and the set of edges, respectively, with $|V| = n$ and $|E| = m$. Edges are observed in arbitrary order through the graph stream $\Sigma = \{e^{(1)}, ..., e^{(m)}\}$. An edge $e^{(t)} = \{u, v\} \in E$ is an unordered pair of nodes. Note that Σ is an *insertion-only* stream: edges can only be added and not deleted. Our algorithm `Tonic` can be adapted to fully dynamic streams with arbitrary edge insertions and deletions, but we present only the variant for insertion-only streams due to space constraints and for simplicity. The description of `Tonic` for fully dynamic streams is in the extended version of this paper [**?**].

For $t = 1, ..., m$, we denote with $G^{(t)} = (V, E^{(t)})$, where $E^{(t)} = \{e^{(1)}, \dots, e^{(t)}\}$, the graph up to time t; note that $G^{(m)} = G$. We say that $\Delta = \{u, v, w\}$ is a *triangle* in $G^{(t)}$ if edges $\{u, v\}, \{u, w\}$, and $\{v, w\}$ all appear in $E^{(t)}$. We also say that a triangle Δ in $G^{(t)}$ is incident to a vertex v if $v \in \Delta$.

For every $t = 1, 2, \dots$, we are interested in approximating the *global* number $T^{(t)}$ of triangles in $G^{(t)}$ as well as the *local* number $T_u^{(t)}$ of triangles incident to u in $G^{(t)}$ for every $u \in V$. In our work, we make the following assumptions: no exact information about the input stream (e.g., the true number of nodes, the true number of edges, the true number of triangles) is available to the algorithm; we can store at most k edges in memory (k is part of the input). We are also going to assume that the edge stream can be only accessed once (i.e., we consider one-pass streaming algorithms) to count triangles. However, we also describe a simple but effective predictor that can be obtained with a fast additional pass on the stream Σ.

A. Waiting Room Sampling

Our algorithm `Tonic` uses waiting room sampling, proposed by Shin et al. [29], that allows to exploit *temporal localities* observed in most real-world datasets. Temporal localities represent the tendency that future edges are more likely to form triangles with recent edges rather than with older edges in real graph streams. To exploit such temporal localities in triangle counting, Shin et al. [29] propose to always store the most recent edges of the stream. More in detail, in waiting room sampling with memory budget k, for some constant $\alpha \in (0,1)$, a portion $k\alpha$ of the memory, called *waiting room* $\mathcal{W}$, is reserved to keep the $k\alpha$ most recent edges from the stream Σ.

B. Predictor

Our algorithm makes use of a *predictor*. We consider predictors that provide some information about the *heaviness* of edges, that is, the number of triangles in which edges are involved. More formally, for an edge $e = \{u, v\}$, let $\Delta(e)$ be the number of triangles containing both u and v (i.e., e is an edge of the triangle). We define a predictor O_H for the heaviness of edges as a function $O_H : E \to \mathbb{R}^+$, where $O_H(e)$ is a measure *related* to $\Delta(e)$: our algorithm uses O_H only to *compare* the heaviness of edges, so $O_H(e)$ does not need to be a prediction for the actual value of $\Delta(e)$. In particular, for every time t, our algorithm `Tonic` uses O_H to keep a fixed size set with the heaviest edges observed up to time t.

The intuition for using such predictions is that in most cases triangles are not distributed equally across edges, thus O_H allows to focus on edges that most contribute to the number of triangles. The impact of O_H then depends on how much the edges it predicts as *heavy* actually contribute to triangle counting (in relation to the other edges).

Note that an heaviness predictor is targeting global counting of triangles, in contrast to the local number of triangles incident to *each* vertex. However, an heaviness predictor is useful also for vertices with *high* local triangles counts, which are often the ones of interest when analyzing local counts.

C. Reservoir Sampling

The last component of our algorithm `Tonic` is the sampling of edges through *reservoir sampling* [37]. In particular, at time t, `Tonic` stores a set of $k' < k$ edges sampled uniformly at random among the *light edges* of $G^{(t)}$, that are edges of $G^{(t)}$ that are neither in the waiting room $\mathcal{W}$ nor kept in the set of (predicted) heavy edges (see Section IV). More in detail, let $L^{(t)}$ be the set of light edges in $G^{(t)}$. At time t, reservoir sampling maintains a sample $\mathcal{S}_L$ of at most k' edges as follows: let $e^{(t)}$ be the edge observed at time t; if $|L^{(t)}| < k'$, then $e^{(t)}$ is added to $\mathcal{S}_L$; otherwise, with probability $k'/|L^{(t)}|$ remove an edge chosen uniformly at random from $\mathcal{S}_L$ and add $e^{(t)}$ to $\mathcal{S}_L$ (with probability $1-k'/|L^{(t)}|$, $\mathcal{S}_L$ does not change).

Let $\mathcal{S}_L^{(t)}$ be the set of edges in $\mathcal{S}_L$ at the end of time step t. Reservoir sampling guarantees [37] that for every time step t with $|L^{(t)}| \geq k'$, if we let A be any subset of $L^{(t)}$ of size $|A| = k'$, then $\mathbb{P}\left[\mathcal{S}_L^{(t)} = A\right] = \frac{1}{\binom{|L^{(t)}|}{k'}}$, that is, $\mathcal{S}_L^{(t)}$ is a uniform sample of size k' from the set $L^{(t)}$ of light edges seen so far in the graph stream.

IV. Tonic: Counting Triangles with Predictions

We now describe our algorithm `Tonic` for counting triangles in graph streams with predictions. Again, due to space constraints and for the sake of clarity, we describe the version for intersertion-only streams (the description of the algorithm for fully-dynamic streams is in [**?**]). Similarly to previous approaches, `Tonic` uses reservoir sampling and waiting room sampling, but in addition it devotes part of its memory to store edges predicted as *heavy* by a predictor. In particular, `Tonic` stores a set $\mathcal{S}$ of at most k edges in memory, where k is provided in input. The set $\mathcal{S}$ comprises three disjoint sets of edges: the waiting room $\mathcal{W}$, storing the $k\alpha$ most recent edges in the stream, where $\alpha \in (0,1)$ is constant; the set $\mathcal{H}$ with the heaviest edges (according to the predictor) observed in the stream, of size $|\mathcal{H}| = k(1-\alpha)\beta$, where $\beta \in (0,1)$ is constant; the set $\mathcal{S}_L$, storing a sample of dimension $k(1-\alpha)(1-\beta)$ of *light* edges, i.e., edges observed in $\Sigma^{(t)}$ that are neither in $\mathcal{W}$ nor in $\mathcal{H}$ at time t . `Tonic` is a 1-pass streaming algorithm, that is, it returns estimates of global and local counts of all the triangles in the graph G at the end of only one pass of edges in the input stream.

As state in Section III-B, `Tonic` employs a predictor O_H that predicts a measure of heaviness for every edge. In general, `Tonic` makes no assumption on the quality of the predictor O_H. For example, we do not assume that O_H provides reliable measure of the number $\Delta(e)$ of triangles that include edge e for all e. Our algorithm provides unbiased estimates independently of the quality of the predictor O_H, and our analysis (see Section IV-A) shows that, as long as the total number of triangles captured by edges kept in $\mathcal{H}$ (due to the predictions of O_H) is large enough (compared to the total number of triangles), the use of O_H leads to improved estimates of the number of triangles. Moreover, we prove that if the predictor O_H makes random predictions, our algorithm does not provide worse estimates than previous approaches.

We now describe `Tonic`; its pseudocode is presented in Algorithm 1. `Tonic` first initializes the sets $\mathcal{W}, \mathcal{H}$, and $\mathcal{S}_L$, the estimate $\hat{T}$ of the global number of triangles, and the observed number ℓ of light edges (lines 1-2). Let t be the instant of time in which the edge $\{u, v\}$ arrives in the stream (line 3). Let $\mathcal{S}^{(t)}$ denote the set of edges present in $\mathcal{S}$ at time t, and analogously $\mathcal{W}^{(t)}$, $\mathcal{H}^{(t)}$ for the waiting room $\mathcal{W}$ and the heavy edges $\mathcal{H}$, respectively. At time t, the algorithm performs the following steps. First, it counts each occurrence of triangles containing $\{u, v\}$ in the set $\mathcal{S}^{(t)}$ (line 4), computing for each triangle $\Delta = \{u, v, w\}$ a corresponding probability p_Δ according to whether $\{u, w\}$ and/or $\{v, w\}$ belong to the sample $\mathcal{S}_L$ of light edges or not; p_Δ is used as a correction factor for updating the estimated counts of the global and local number of triangles. `Tonic` then updates $\mathcal{W}$, $\mathcal{H}$, and $\mathcal{S}_L$ as follows. If $\mathcal{W}$ is not full, it inserts $\{u, v\}$ in $\mathcal{W}$ (line 5). If $\mathcal{W}$ is full, it replaces the oldest edge $\{x, y\}$ in $\mathcal{W}$ with $\{u, v\}$ (lines 7-8), and, if $\mathcal{H}$ is not full, it inserts $\{x, y\}$ in $\mathcal{H}$ (lines 9-10). Otherwise, we are going to observe a light edge and increment counter ℓ (line 12). Let e' be the *lightest* edge in $\mathcal{H}$, i.e., $e' = \arg\min_{e \in \mathcal{H}} O_H(e)$ (line 13). Let $e_{\max}$ be the heaviest edge between $\{x, y\}$ and e', and let $e_{\min}$ be the lightest edge between such edges. Then `Tonic` keeps $e_{\max}$ in $\mathcal{H}$ and updates $\mathcal{S}_L$: if $\mathcal{S}_L$ is not full, it inserts $e_{\min}$ in $\mathcal{S}_L$; otherwise, it updates $\mathcal{S}_L$ using reservoir sampling for edge $e_{\min}$ (lines 14-19). Tonic reports the final estimates $\hat{T}$ and $\hat{T}_u$ at the end of the stream (line 20).

Algorithm 1: Tonic $(\Sigma, k, \alpha, \beta, O_H)$

```
Input  : Arbitrary order edge stream $\Sigma = \{e^{(1)}, e^{(2)}, ...\}$;
         memory budget $k$; fraction of waiting room space
         $\alpha$; fraction of heavy edges space $\beta$; edge heaviness
         predictor $O_H$
Output : Estimate of global triangles count $\hat{T}$; estimate of
         local triangles count $\hat{T}_u$ for each node $u$
1  $\mathcal{W} \longleftarrow \emptyset$; $\mathcal{H} \longleftarrow \emptyset$; $\mathcal{S}_L \longleftarrow \emptyset$;
2  $\hat{T} \longleftarrow 0$; $\ell \longleftarrow 0$;
3  for each edge $e^{(t)} = \{u, v\}$ in the stream $\Sigma$ do
4      CountTriangles($\{u, v\}, \mathcal{W} \cup \mathcal{H} \cup \mathcal{S}_L, \ell$);
5      if $|\mathcal{W}| < k\alpha$ then $\mathcal{W} \longleftarrow \mathcal{W} \cup \{\{u, v\}\}$;
6      else
7          $\{x, y\} \longleftarrow$ oldest edge in $\mathcal{W}$;
8          $\mathcal{W} \longleftarrow \mathcal{W} \setminus \{\{x, y\}\} \cup \{\{u, v\}\}$;
9          if $|\mathcal{H}| < k(1-\alpha)\beta$ then
10             $\mathcal{H} \longleftarrow \mathcal{H} \cup \{\{x, y\}\}$;
11         else
12             $\ell \longleftarrow \ell + 1$;
13             $\{u', v'\} \longleftarrow$ lightest edge in $\mathcal{H}$;
14             if $O_H(\{x, y\}) > O_H(\{u', v'\})$ then
15                 $\mathcal{H} \longleftarrow \mathcal{H} \cup \{\{x, y\}\} \setminus \{\{u', v'\}\}$;
16             else $\{u', v'\} \longleftarrow \{x, y\}$; ;
17             if $\ell < k(1-\alpha)(1-\beta)$ then
18                 $\mathcal{S}_L \longleftarrow \mathcal{S}_L \cup \{\{u', v'\}\}$;
19             else SampleLightEdge($\{u', v'\}$, $\mathcal{S}_L, \ell$);
20 return $\hat{T}$, $\hat{T}_u$ for each node $u \in \mathcal{S} = \mathcal{W} \cup \mathcal{H} \cup \mathcal{S}_L$;
```

At any time t, $\hat{T}$ provides an estimate of the (global) number of triangles observed in the graph $E^{(t)}$ up to time t. For local triangles, Tonic maintains estimates $\hat{T}_u > 0$ for some vertices u in V, in particular for vertices with triangles contributing to $\hat{T}$; for all other vertices, the estimated number of triangles is 0.

Tonic makes use of two subroutines, CountTriangles and SampleLightEdge. CountTriangles (Algorithm 2) counts the number of triangles *closed* by the current edge in the stream, computes the probability that such triangle is sampled by the algorithm and uses such probability to update the relevant counts (see [**?**] for the correctness of the probabilities computed by CountTriangles). SampleLightEdge (Algorithm 3) uses reservoir sampling (see Section III-C) to update the sample $\mathcal{S}_L$ of the set L of light edges observed in stream. Also, we assume that *FlipBiasedCoin(p)* flips *HEAD* with probability p.

A. Analysis

In this section, we analyze our algorithm Tonic. In particular, we first prove that Tonic provides unbiased estimates of the number of global/local triangles at each time step. We then analyze the time complexity of our algorithm, and also analytically assess when the use of a noisy predictor leads to estimates with smaller variance (i.e, that are more accurate) compared to the WRS algorithm from [29]. For lack of space, all proofs and analyses, including those for our algorithm for fully dynamic streams, are in the extended version [**?**].

Algorithm 2: CountTriangles($\{u, v\}, \mathcal{S}, \ell$)

```
Input : edge $\{u, v\}$; subgraph $\mathcal{S} = \left(\hat{V},\ \hat{E}\right)$; number of
        predicted light edges $\ell$
1  for each node $w$ in $\mathcal{N}_u \cap \mathcal{N}_v$ do
2      initialize $\hat{T}_u$, $\hat{T}_v$, $\hat{T}_w$ to zero if not set yet;
3      $p_{uvw} = 1$;
4      if $\{w, u\} \in \mathcal{S}_L$ AND $\{v, w\} \in \mathcal{S}_L$ then
5          $p_{uvw} = \min\left(1, \frac{k(1-\alpha)(1-\beta)}{\ell} \times \frac{k(1-\alpha)(1-\beta)-1}{\ell-1}\right)$;
6      else if $\{w, u\} \in \mathcal{S}_L$ OR $\{v, w\} \in \mathcal{S}_L$ then
7          $p_{uvw} = \min\left(1, \frac{k(1-\alpha)(1-\beta)}{\ell}\right)$;
8      increment $\hat{T}$, $\hat{T}_u$, $\hat{T}_v$, $\hat{T}_w$ by $1/p_{uvw}$;
```

Algorithm 3: SampleLightEdge($\{u, v\}, \mathcal{S}_L, \ell$)

```
Input : edge $\{u, v\}$; current sample $\mathcal{S}_L$ of light edges;
        number of observed light edges $\ell$ in the stream
1  $p_{sampling} = \frac{k(1-\alpha)(1-\beta)}{\ell}$;
2  if FlipBiasedCoin($p_{sampling}$) == HEAD then
3      $\{\bar{u}, \bar{v}\} \longleftarrow$ edge sampled uniformly at random from $\mathcal{S}_L$;
4      $\mathcal{S}_L \longleftarrow \mathcal{S}_L \setminus \{\{\bar{u}, \bar{v}\}\} \cup \{\{u, v\}\}$;
```

1) Unbiasedness: the unbiasedness of the estimates reported by Tonic is provided by the following result.

Theorem IV.1. *Let $T^{(t)}$ and $T_u^{(t)}$ be the true global count of triangles in the graph and the true local triangle count for node $u \in V$ at time t, respectively. We have:*

$$\mathbb{E}\left[\hat{T}^{(t)}\right] = T^{(t)},\ \forall\ t \geq 0$$

$$\mathbb{E}\left[\hat{T}_u^{(t)}\right] = T_u^{(t)}\ \forall u \in V,\ \forall\ t \geq 0$$

2) Time Complexity: in the following, we analyze time complexity of Tonic. In the analysis we also account for how the sets of edges stored by Tonic are implemented. The worst-case time complexity of Tonic is dominated by computing the number of triangles the last observed edge closes, plus a logarithmic factor due to retrieving, for each edge in the graph stream, the lightest edge from the set of $\mathcal{H}$.

Theorem IV.2. *Given an input graph stream Σ of insertion-only edges, and given $\alpha = |\mathcal{W}|/k$, $\beta = |\mathcal{H}|/k(1-\alpha)$, Tonic processes each edge in Σ in $\mathcal{O}(k + \log(k(1-\alpha)\beta))$ time.*

From the above, the total time to compute the estimation of local and global triangles for the entire graph stream of m edges is $\mathcal{O}(mk + m\log(k(1-\alpha)\beta))$. This is a worst-case bound since in practice, in real graph streams, the complexity of the computation of common neighbors for nodes u and v is much smaller than k, i.e., our memory budget.

3) Comparison with WRS for global triangles count: we now prove that our algorithm leads to better estimates than WRS [29] for the global number of triangles when the predictions from the predictor O_H are *useful*, in the sense that they lead to consider as *heavy*, edges that are involved in a large number of triangles. Note that this is different from having a *perfect oracle* (i.e., making no errors).

For the sake of simplicity, in the analysis we consider a simplified version of the WRS algorithm and a simplified version of Tonic algorithm that capture the main features of the two approaches. The simplified version of the WRS samples each light edge (i.e, that leaves the waiting room) independently with probability p; the simplified version of Tonic uses a predictor that predicts an edge leaving the waiting room as *heavy* or *light*, keeps (predicted) heavy edges in $\mathcal{H}$, and samples each light edge (see line 19 of Algorithm 1) with probability $p' < p$, where p is the probability that an edge is sampled by WRS. Note that by properly fixing p', accounting for the memory budget for heavy edges (i.e., $k\,(1-\alpha)\,\beta$), the two algorithms have the same expected memory budget. We assume that the waiting room has the same size in both WRS and Tonic. Note that triangles where the first two edges are in the waiting room $\mathcal{W}$ at the time of discovery are counted (with probability 1) by both algorithms. Since the size of the waiting room is the same, the sets of triangles counted (with probability 1) in the waiting room by the two algorithms are identical. Given that we are interested in the difference in the estimates from the two algorithms, we exclude such triangles from our analysis (i.e., all quantities below are meant after discarding such triangles).

Note that the quality of the approximation reported by our algorithm Tonic w.r.t. WRS must depend on several quantities: the number of triangles in which heavy edges are involved; the number of triangles in which light edges are involved; p and p', that govern the memory allocated for light edges; the quality of the predictor in differentiating heavy edges from light edges. Our result provides a formal relation between such quantities.

Let $\Delta(e)$ be the number of triangles in which edge e is involved. Let $T^H = \sum_{h\in\mathcal{H}} \Delta(h)$ be the sum, over heavy edges, of the number of triangles in which heavy edges appear; analogously, let $T^L = \sum_{l\in L} \Delta(l)$ be the sum, over light edges, of the number of triangles in which light edges appear.

In our analysis we make some assumptions on the predictor; in particular, we will assume heavy edges appear in $\geq \rho$ triangles, while light edges appear in $< \rho$ triangles, for some $\rho \geq 3$. However, to capture the errors of the predictor, in particular around the threshold ρ, we assume that edges h predicted as heavy by the predictor are involved in $\Delta(h) \geq \rho/c$ triangles, while edges l predicted as light by the predictor are involved in $\Delta(l) < c\rho$ triangles, for some constant $c \geq 1$. Note that for edges e with $\rho/c < \Delta(e) < c\rho$ the predictor can make arbitrarily wrong predictions.

Proposition 1. *Let* $\mathbf{Var}[\hat{T}_{\texttt{WRS}}(p)]$ *be the variance of the estimate* $\hat{T}_{\texttt{WRS}}$ *obtained by* WRS *when light edges are sampled independently with probability* p*, and let* $\mathbf{Var}[\hat{T}_{\texttt{Tonic}}(p')]$ *be the variance of the estimate* $\hat{T}_{\texttt{Tonic}}$ *obtained by* Tonic *when light edges are sampled with probability* p'*. Then* $\mathbf{Var}[\hat{T}_{\texttt{Tonic}}(p')] \leq \mathbf{Var}[\hat{T}_{\texttt{WRS}}(p)]$ *if*

$$\frac{T^H}{T^L} > 3\frac{(1/p'^2 - 1/p^2) + c\rho(1/p' - 1/p)}{(1/p-1)(3+4\rho/c)}.$$

The result above explicits a trade-off between the quality of the predictor (represented by c), the impact of heavy edges in the count (represented by ρ and T^H) with respect to light edges (represented by T^L), and the fraction of memory allocated for heavy edges in our algorithm (that depends on the difference between p and p'.) For example, if $p = 0.1$ and $p' = 0.09$ (these are representative values for our experimental evaluation), $c = 1.5$, and $\rho = 10$, than the bound above is $\frac{T^H}{T^L} > 0.45$, that corresponds to the sum of the counts for heavy edges being at least one third of all triangles (that do not have two edges in the waiting room $\mathcal{W}$ when counted); if instead the parameters are as above but $\rho = 100$, then $\frac{T^H}{T^L} > 0.24$, that corresponds to the sum of the counts for heavy edges being at least one fifth of all triangles (again, that do not have two edges in the waiting room $\mathcal{W}$ when counted). For the latter case, note that $c = 1.5$ implies that the predictor is not very accurate: a light edge l (with $\Delta(l) < 100$ by definition) may be mispredicted as heavy as long as $\Delta(l) > 66$, while an heavy edge h (with $\Delta(h) \geq 100$ by definition) may be mispredicted as light edge as long as $\Delta(h) < 150$.

The result above formalizes the intuition that the predictor helps when it provides fairly reliable information on heavy edges. The following result instead proves that, when the predictor does not provide useful information about heavy edges, our algorithm provides estimates as accurate as WRS, in the sense that the variances of the estimates from the two algorithms are the same when the same amount of memory is used.

Proposition 2. *If the predictor* O_H *predicts a random set of edges as heavy edges, and* WRS *and* Tonic *use the same amount of memory, then* $\mathbf{Var}[\hat{T}_{\texttt{Tonic}}(p')] = \mathbf{Var}[\hat{T}_{\texttt{WRS}}(p)]$.

Strikingly, our experimental evaluation shows that our algorithm Tonic provides estimates of quality similar to WRS even when the most *adversarial* predictor is provided (see the extended version [**?**]).

B. A Simple Domain-Independent Predictor

The predictor O_H used by Tonic could be implemented by using one of the several machine learning models that may consider information other than the graph G in its prediction. For example, in social networks, information about the users may be useful to predict the heaviness of an edge. In protein interaction networks, the function and properties of the protein provide a lot of information on its heaviness. However, we now describe a simple predictor based only on the graph structure that, as we will show, is extremely powerful in practice.

We define a simple predictor MinDegreePredictor that predicts, as measure of heaviness for $\{u, v\}$, the minimum between the degree $deg(u)$ of u and the degree $deg(v)$ of v, that is $O_H = \min\{deg(u), deg(v)\}$. Note that the degree $deg(u)$ for each node $u \in V$ can be computed easily with 1 pass on the data whenever the whole stream is available beforehand. Moreover, in practical applications, one can measure the number of observed edges involving a given node

u in a *training* phase, and then use such information for the prediction in later phases. Additionally, to reduce the memory required for the predictor, instead of storing *all* nodes degrees, one can simply store the largest ones, and assume a value of 0 for the degree of all other nodes.

V. Experimental Evaluation

In this section we present the results of our experiments. Due to space constraints, we only present a subset of results for the estimation of the global number of triangles for insertion-only streams. The complete results for estimating the local number of triangles, and for fully dynamic streams are in the extended version of the paper [**?**]. The goals of our experiments are: i) to assess the dependency of `Tonic` from the relative dimension of the waiting room (parameter α), from the relative dimension of the heavy edges set ($(1-\alpha)\beta$), and from the total memory budget (k); ii) to assess the improvement in the estimates that results from the predictor; iii) to compare `Tonic` with state-of-the-art algorithms (with and without predictors) for counting triangles in edge streams on single streams and on sequences of edge streams, where the predictor is learned only in the first stream (i.e., graph) of the sequence; iv) to assess the quality of `Tonic`'s estimates during the evolution of an input graph stream.

Experimental Setup. We implemented `Tonic` in C++17. The code to reproduce all experiments is available anonymously at https://anonymous.4open.science/r/Tonic-F2C4/. All the code was compiled under `gcc` 9.4.0 and ran on a 2.20 GHz Intel Xeon CPU with 1 TB of RAM, on Ubuntu 20.04. If not explicitly specified, we fixed the memory budget of each algorithm to $k = m/10$ and for each run we report mean and standard deviation across 50 independent trials. To measure accuracy for global triangles estimates, we considered the *global relative error* at the end of the stream, that is $|\hat{T} - T|/T$.

Datasets. We considered both single graph and sequence of graph streams as datasets of interest, representing social and citation networks, and autonomous system (AS) relationships, downloaded from [13], [17], [28]. Datasets' names and statistics are summarized in Table I. A complete description of the considered dataset can be found in the extended version of the paper [**?**]. From each dataset, we removed self-loops and multiple edges, deriving a stream of insertion-only edges, for consistency with previous works. For graph sequences (i.e., the last four rows of the table), the statistics in Table I refer to the graph with highest number of nodes.

Predictors. For our algorithm `Tonic`, the predictors used depend on whether we analyze a single graph stream or a sequence of graph streams. For single streams, we considered three predictors: `OracleExact`, where $O_H(e)$ is the number $\Delta(e)$ of triangles involving e in the graph stream (i.e., the heaviness); `Oracle-noWR`, where $O_H(e)$ is obtained subtracting to $\Delta(e)$ the number of triangles for which e is in the waiting room $\mathcal{W}$; `MinDegreePredictor`, the predictor described in Section III-B for which $O_H(\{u,v\})$ is the minimum between the degree of u and the degree of v. Note that, when analyzing single streams, `OracleExact` and `Oracle-noWR` are mostly *ideal* and not *practical* predictors (since they require to solve the counting problem exactly first), but we use them to study the potential gain obtained with a predictor. In addition, `OracleExact` represents a general predictor for heaviness, while `Oracle-noWR` is a predictor tied to the counting strategy employed by our algorithm. In contrast, `MinDegreePredictor` is a predictor that can be easily implemented with a single pass on the edge stream, and is therefore practical when the entire edge stream is available beforehand. For `OracleExact` and `Oracle-noWR`, the predictor maintains the top 10% edges sorted by predicted values (i.e., $O_H(e)$). Hence, we consider predictors (practical or ideal) that provide accurate information only for the top 10% edges, while for the remaining 90% of edges it outputs $O_H(e) = 0$. For `MinDegreePredictor`, we use a different representation based on nodes. In particular, the predictor stores $\bar{n}$ pairs $(u; deg(u))$ corresponding to the $\bar{n}$ highest degree nodes in the graph. For an edge $e = \{u,v\}$, `MinDegreePredictor` produces in output $\min(deg(u), deg(v))$ if both $(u; deg(u))$ and $(v; deg(v))$ are stored in the predictor, and 0 otherwise. The value $\bar{n}$ is fixed so to correspond to the number of (unique) nodes that would be required to compute the exact value for the top 10% edges (according to the `MinDegreePredictor`); note that the nodes stored by the predictor may actually be different from the ones required to predict the exact (min-degree) value for the top 10% edges, since we are storing the highest degree nodes. Further details can be found in [**?**].

For sequences $\Sigma_1, \Sigma_2, \dots$ of graph streams, we considered the same predictors, but *trained* using only the *first* stream Σ_1. For example, in `OracleExact`, $O_H(e)$ is the number of triangles of Σ_1 in which e appears; in later streams, edges e adjacent to vertices not in Σ_1 always have $O_H(e) = 0$. Note that `OracleExact`/`Oracle-noWR` can be obtained by solving the problem exactly on the graph stream Σ_1, and are therefore *practical* whenever this is feasible.

Baselines. We compared our algorithm `Tonic` with

TABLE I
Datasets' statistics: number n of nodes; number m of edges; number T of triangles

Dataset	n	m	T
Single Graphs			
Edit EN Wikibooks	$133k$	$386k$	$178k$
SOC YouTube Growth	$3.2M$	$9.3M$	$12.3M$
Cit US Patents	$3.7M$	$16.5M$	$7.5M$
Actors Collaborations	$382k$	$15M$	$346.8M$
Stackoverflow	$2.5M$	$28.1M$	$114.2M$
SOC LiveJournal	$4.8M$	$42.8M$	$285.7M$
Twitter-merged	$41M$	$1.2B$	$34.8B$
Snapshot Sequences			
Oregon (9 graphs)	$11k$	$23k$	$19.8k$
AS-CAIDA (122 graphs)	$26k$	$53k$	$36.3k$
AS-733 (733 graphs)	$6k$	$13k$	$6.5k$
Twitter (4 graphs)	$29.9M$	$373M$	$4.4B$

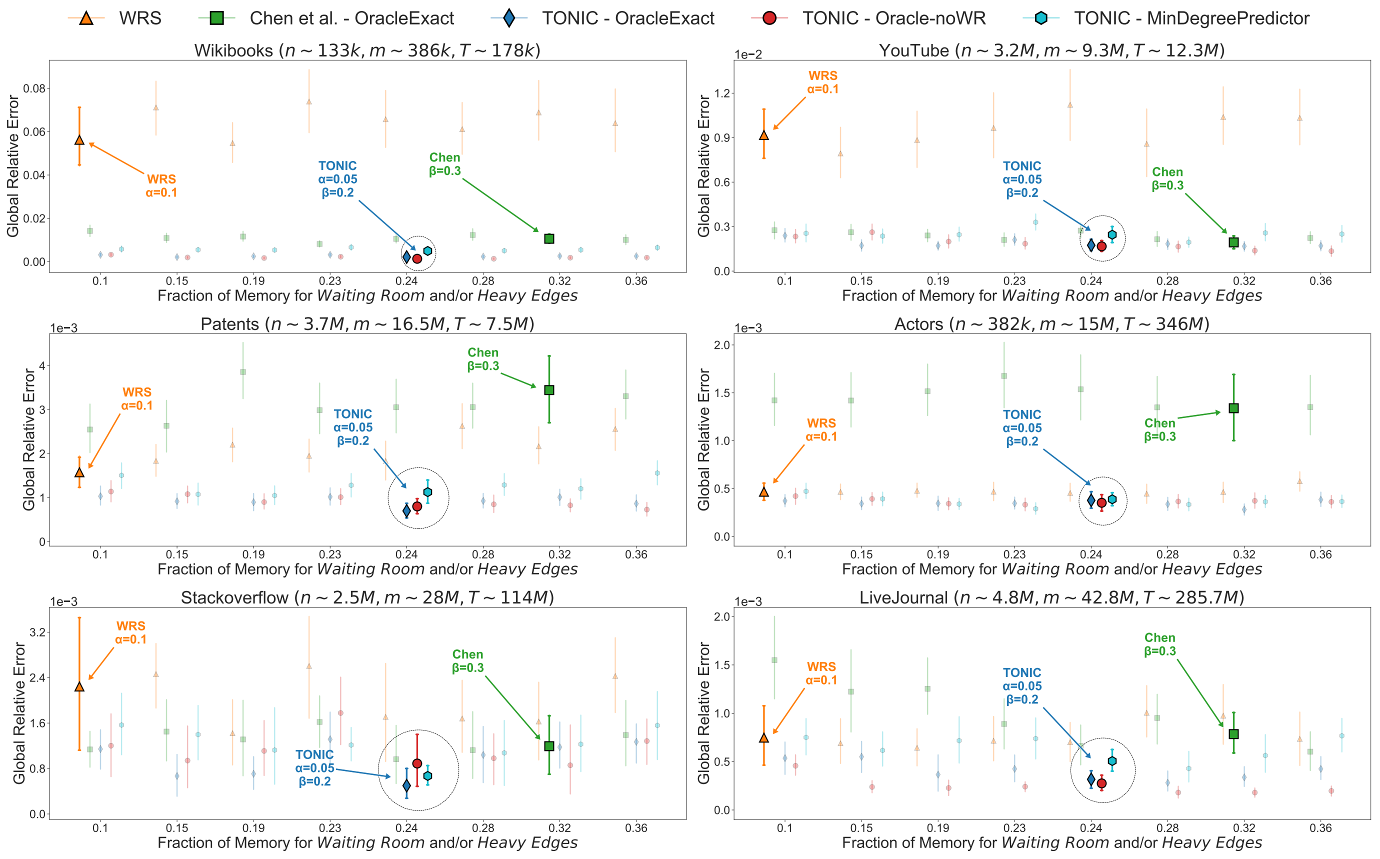
Fig. 1. Error vs fraction of memory budget used for waiting room and/or heavy edges. For each combination of algorithm and parameter (including predictor for `Tonic`), the average and 95% confidence interval over 50 repetitions are shown. The chosen configuration for `Tonic` and the configurations suggested by `WRS` and `Chen` publications are highlighted.

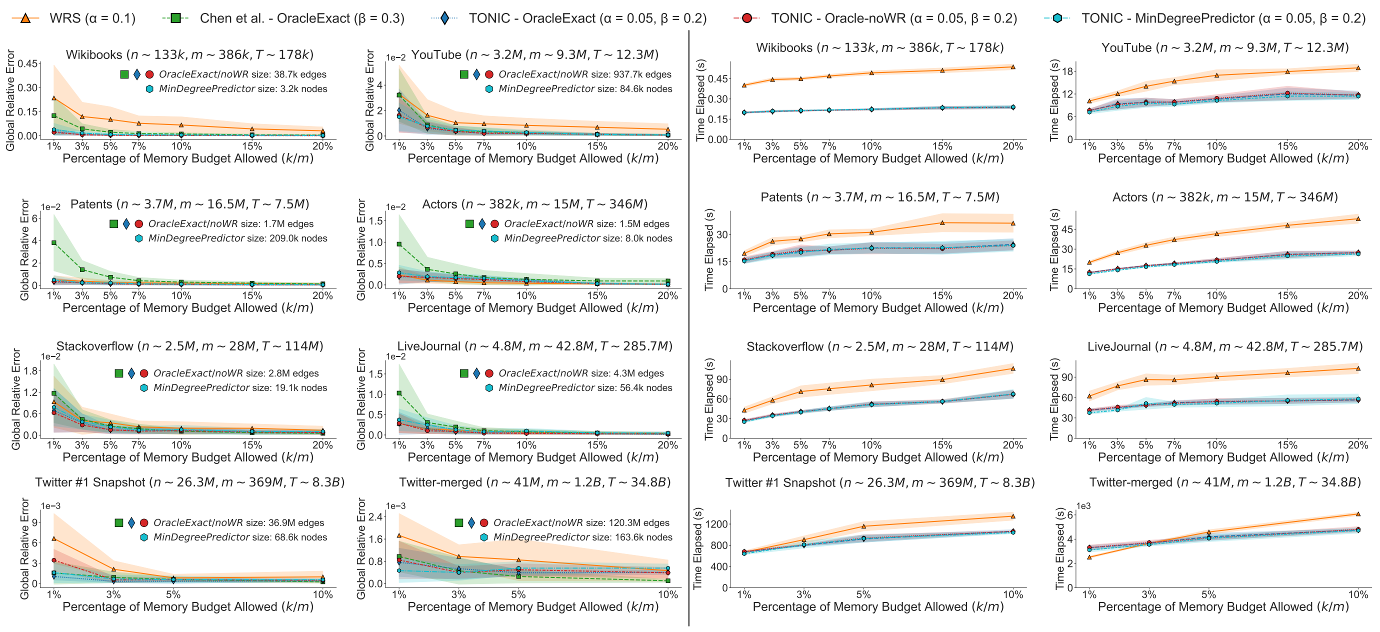
Fig. 2. Error (left) and runtime (right) vs memory budget. `Chen` runtimes are not shown for clarity, since they are 4-15 times bigger than `Tonic` runtime. For each combination of algorithm and parameter (including predictor for `Tonic`), the average and standard deviation over 50 repetitions are shown. The algorithms parameters are as in legend (for `WRS` and `Chen` they are fixed as in the respective publications; for `Tonic` they are as chosen in Fig. 1).

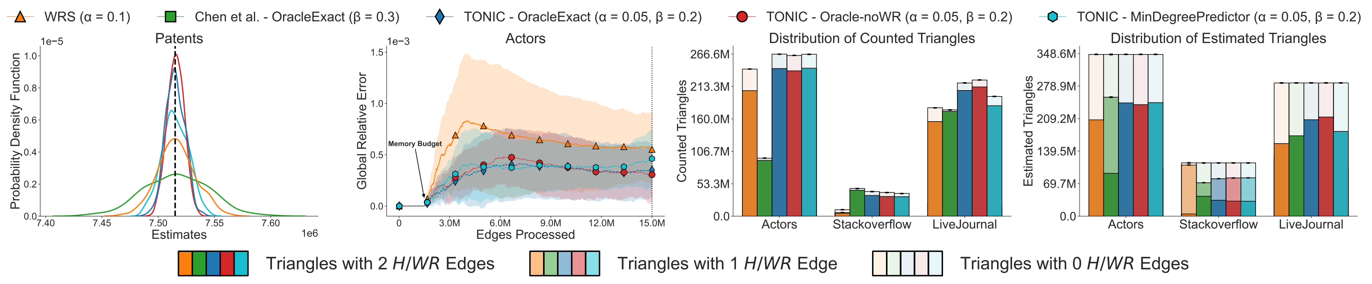

Fig. 3. (Left) Distribution of estimates for Patents dataset. (Center-left) Estimation error as time progresses on Actors dataset. (Center-right) Number and type of triangles counted by each algorithm on three datasets. (Right) Fraction of each type of triangles in the total estimates by each algorithm on three datasets.

WRS [16], [29] (considering the algorithm for insertion-only graph streams), that is the state-of-the-art for global and local triangles in (insertion-only) graph streams, and with the algorithm from [6] (considering the algorithm for arbitrary order streams), that we denote as Chen, which is the only algorithm that uses predictors and is limited to estimating global triangles at the end of the (insertion-only) stream[1]. For fully-dynamic streams, we compared our algorithm Tonic-FD with WRS_{del} [16] (considering the version for FD streams) and with $\text{ThinkD}_{\text{acc}}$ (the algorithm with fixed memory from [30], [31]), which are both state-of-the-art for global and local triangles in (FD) graph streams. In all experiments, all algorithms are provided with the same memory budget. The main parameter of WRS and WRS_{del} is the fraction α of the memory allocated to the waiting room, while the main parameter for Chen is the fraction β of the memory allocated to the heavy edges. For Chen we used as predictor OracleExact, that is, for single streams we are considering the best predictor that it could have access to. $\text{ThinkD}_{\text{acc}}$ has no tunable parameters.

Results. First, we ran Tonic for various values of the parameters α and β (see extended version [**?**] for more details). We also ran WRS and Chen with various values of α (for WRS) and β (for Chen). Fig. 1 shows the error as a function of the fraction of memory budget used for the waiting room and/or heavy edges ($\alpha + (1-\alpha)\beta$ for Tonic, α for WRS, and β for Chen) for all our single graphs but the Twitter ones, due to time impracticability on such large graphs. We observe that WRS and Chen have performance that strongly depends on the dataset and the parameters, while our algorithm Tonic provides estimates with low error for all values of α and β, with the best combination depending on the dataset, and for all datasets. However, overall the combination $\alpha = 0.05$ and $\beta = 0.2$ leads to good results across all datasets, and in all subsequent experiments we fixed the parameters to such values. Note that Tonic with $\alpha = 0.05$ and $\beta = 0.2$ is always better than WRS and Chen with parameters as suggested in the respective publications ($\alpha = 0.1$ and $\beta = 0.3$). Interestingly, Tonic with the MinDegreePredictor (the only practical one) is always close to Tonic with predictors OracleExact and Oracle-noWR, and always outperforms WRS and Chen other than for Youtube, where Chen (empowered by the OracleExact) shows slightly lower error. These results show that our algorithm Tonic is robust to the choice of the parameters α and β, and provides accurate estimates consistently across all datasets even when a simple practical predictor is used, adapting to the data distribution thanks to the use of the predictor.

We then assessed how Tonic behaves in terms of error and runtime as a function of the memory budget k, and compared it with WRS and Chen. For every dataset, we ran all algorithms with memory budget $k = fm$, with $m = |E|$ and for values of f showed on the x-axis. Fig. 2 (left) shows the estimation errors, with Tonic achieving a better accuracy in almost every case. Fig. 2 (right) shows the runtime only for Tonic and WRS; the runtime of Chen is always at least 4, and up to 15, times larger than Tonic runtime and therefore not shown in Fig. 2 (right) for the sake of clarity. We see that Tonic is always faster than WRS (excluding $1\%m$ memory budget for Twitter-merged dataset, for which the runtime is slightly higher, but still reasonable, while Tonic significantly outperforms WRS in terms of approximation error), showing a much milder slope and hence, in practice, able to scale better with respect to worst-case analyses of the running time. For larger memory budgets, Tonic usually outperforms WRS in terms of approximation error (or is at least comparable to it), while being faster. Therefore, in all scenarios Tonic provides an advantage over WRS. On Twitter-merged for memory budgets $k = 5\%m$ and $k = 10\%m$, Chen has the best accuracy, slightly better than Tonic. This is due to Twitter streams being in adjacency list order, since in such cases the waiting room is not beneficial. However, while the accuracy of Tonic is worse than, but comparable to, Chen, the difference in runtime is huge: for a single run with $k = 10\%m$, Tonic takes ~ 70 minutes, Chen requires more than 16 hours.

Moreover, we assessed the quality of the approximations in terms of unbiasedness, variance, quality of estimates at any time of the stream, and number of counted and estimated triangles. Fig. 3 shows the results for some representative datasets (see [**?**] for the all results). We observe that, as expected, all three algorithms return unbiased estimates, but Tonic has a much lower variance (Fig. 3 left), confirming

[1]While the algorithm described in [6] uses fixed probability sampling, the implementation provided by the authors enforces a maximum memory budget by essentially removing a random light edge from main memory, that does not provide guarantees on the resulting sample.

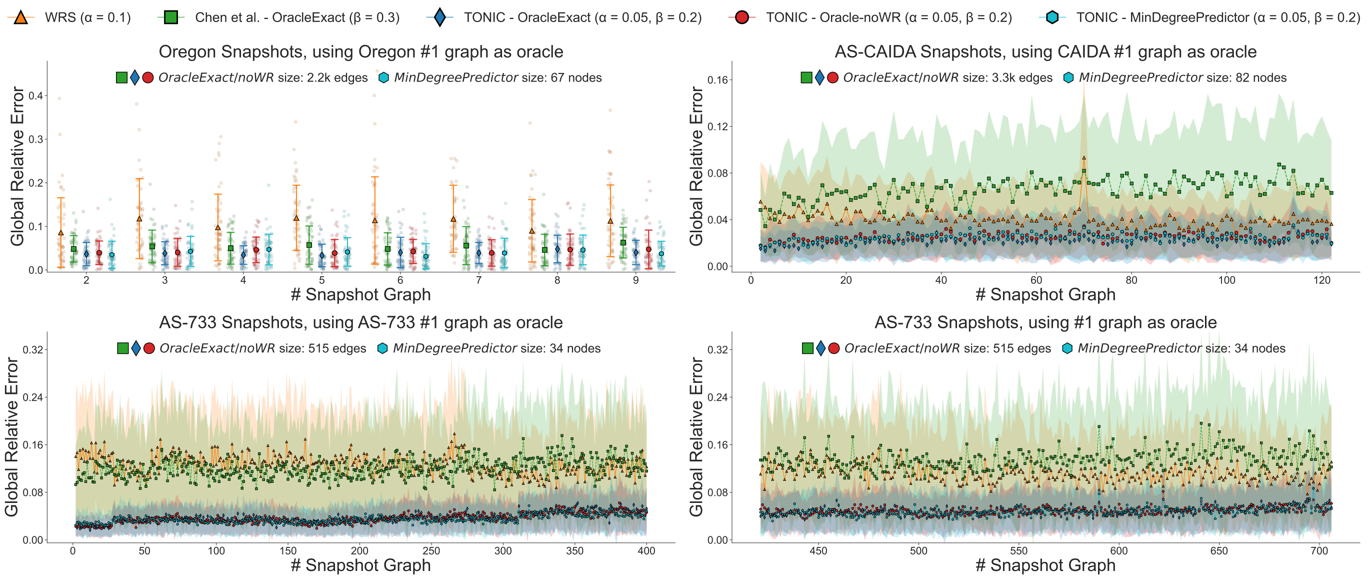

Fig. 4. Error with snapshot networks with sequence of graph streams. The bottom plots are for the first 400 streams (left) and the remaining streams (right) of AS-733. In all cases the predictors are trained only on the first graph stream of the sequence (with results not shown on such graph stream). For each combination of algorithm and parameter (including predictor for `Tonic`), the average and standard deviation over 50 repetitions are shown. The algorithms parameters are as in legend (for `WRS` and `Chen` they are fixed as in the respective publications; for `Tonic` they are as chosen in Fig. 1).

our theoretical analysis (Section IV-A3). We also note that `Tonic` returns more accurate estimates than `WRS` at any time t in the stream (Fig. 3 center-left). Finally, from the number of triangles with 0/1/2 light edges counted and estimated by each algorithm (Fig. 3 center-right, right), we observe that `Tonic` leverages both the waiting room and the predictor, leading to an higher number of discovered triangles with low variance (recall that the fraction of memory budget dedicated to "heavy" edges is 0.3 for `Chen` and 0.19 for `Tonic`).

As final experiments for insertion-only streams, we considered *snapshot networks* with a sequence of graph streams in order to evaluate our algorithm `Tonic` in a more challenging and realistic scenario. We considered datasets Oregon, AS-CAIDA and AS-733, including, respectively, 9, 122, and 733 graph streams, and ran each algorithm on each stream of the sequence, with a memory budget equal to 10% of the number of edges of each considered stream. The predictors used by `Tonic` and `Chen` are trained *only* with the *first* graph stream, and their predictions are used for each subsequent graph. Note that in this case, for each subsequent graph stream, `OracleExact` and `Oracle-noWR` are *imperfect* predictors. Fig. 4 reports the error for each graph stream in the sequence. `Tonic` achieves outstanding performances with all predictors for all three datasets, with errors that are significantly smaller than `WRS` and `Chen` across all graph streams. For AS-CAIDA and AS-733, with hundreds of graph streams, we observe that the results slightly deteriorate as more streams are considered, due to the fact that later graph streams can be very different from the first one in which the predictor was trained. In particular, for later graphs the data is growing significantly: in some cases, nodes, edges, and number of triangles are almost doubled. These results highlight the usefulness and practical impact of using the learned information from the predictor, as done by our algorithm `Tonic`.

Finally, we give a glance of experimental results when dealing with fully dynamic (FD) streams, i.e., with graph streams that allow both insertions and deletions of edges (see [**?**] for all results). More specifically, we create FD streams starting from our 4 snapshot sequences datasets: Oregon, AS-CAIDA, AS-733 and Twitter. For each dataset, starting from the first graph of the sequence, we compute edge additions and removals between the current and the next snapshot. Edge insertions are added to the FD stream by preserving the temporal ordering following the graph sequence, and edge deletions are added to the FD stream with random timestamps inside the time window of the snapshots that we are considering. In the following, we refer to our algorithm for FD streams as `Tonic-FD` for which we describe the general workflow in [**?**]. The oracles and predictors for `Tonic-FD` are trained *only* on the *first* snapshot of the sequence, in the same setting we described for the above snapshot experiments. Fig. 5 shows the estimation error as time progresses during Oregon FD stream. For all algorithms the memory budget is $k = m_{max}/10$ (highlighted by the black arrow in the stream). Note that `Tonic-FD` always achieves the most accurate estimate at any time through Oregon FD stream. Moreover, as shown in the extended version [**?**], our algorithm `Tonic-FD` is always faster than $\texttt{WRS}_{\texttt{del}}$, and is faster than or comparable to $\texttt{ThinkD}_{\texttt{acc}}$, despite requiring the removal of edges within our waiting room and heavy edge set.

VI. Conclusion

In this paper we presented `Tonic`, the first practical algorithm with predictions for fast and accurate triangle counting

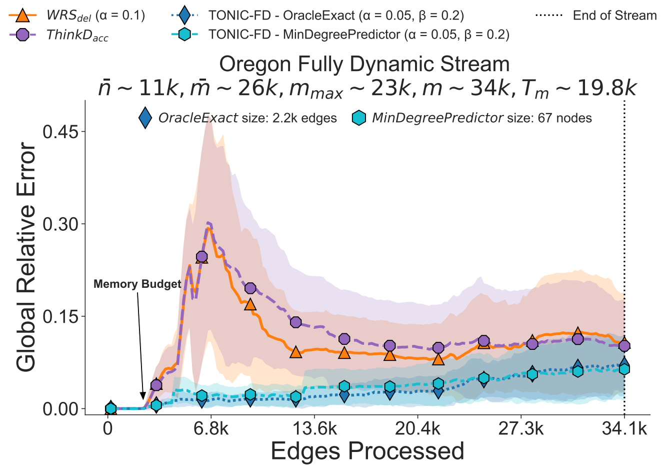

Fig. 5. Estimation error as time progresses during Oregon fully dynamic stream. For each combination of algorithm and parameter (including predictor for Tonic-FD), the average and standard deviation over 50 repetitions are shown. The algorithms parameters are as in legend (for WRS they are fixed as in the respective publication; for Tonic-FD they are as chosen in Fig. 1). $\bar{n}$: number of unique nodes; $\bar{m}$: number of unique edges; m_{max}: maximum number of edges at some time; m: total number of edges; T_m: number of global triangles at the end, derived from the FD stream.

in insertion-only and fully dynamic graph streams. Tonic combines waiting room sampling and reservoir sampling with a predictor for the heaviness of edges. We also propose a simple application-independent predictor, based on the degree of the nodes, that can be efficiently computed with 1 pass when the whole stream is available beforehand, and can be easily computed in a training phase in a sequence of graph streams. Our analysis shows that the predictor leads to improved estimates when the edges predicted as heavy do provide useful information, even if the predictor is far from perfect. Our experimental evaluation shows that Tonic significantly improves the state-of-the-art in terms of error and runtime of the estimates. The improvement is particularly significant in challenging practical scenarios where sequences of hundreds of graph streams are analyzed. Our work opens several directions for future research, including the development of improved predictors.

References

[1] M. Al Hasan and V. S. Dave, "Triangle counting in large networks: a review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2018.

[2] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis, "Efficient algorithms for large-scale local triangle counting," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2010.

[3] J. W. Berry, B. Hendrickson, R. A. LaViolette, and C. A. Phillips, "Tolerating the community detection resolution limit with edge weighting," *Physical Review E*, 2011.

[4] P. Boldi, M. Rosa, M. Santini, and S. Vigna, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," in *20th international conference on World Wide Web*, 2011.

[5] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler, "Counting triangles in data streams," in *25th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2006.

[6] J. Y. Chen, T. Eden, P. Indyk, H. Lin, S. Narayanan, R. Rubinfeld, S. Silwal, T. Wagner, D. P. Woodruff, and M. Zhang, "Triangle and four cycle counting with predictions in graph streams," *arXiv*, 2022.

[7] R. Gemulla, W. Lehner, and P. J. Haas, "Maintaining bounded-size sample synopses of evolving datasets," *The VLDB Journal*, 2008.

[8] B. H. Hall, A. B. Jaffe, and M. Trajtenberg, "The nber patent citation data file: Lessons, insights and methodological tools," 2001.

[9] C.-Y. Hsu, P. Indyk, D. Katabi, and A. Vakilian, "Learning-based frequency estimation algorithms." in *International Conference on Learning Representations*, 2019.

[10] M. Jha, C. Seshadhri, and A. Pinar, "A space efficient streaming algorithm for triangle counting using the birthday paradox," in *19th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, 2013.

[11] D. M. Kane, K. Mehlhorn, T. Sauerwald, and H. Sun, "Counting arbitrary subgraphs in data streams," in *Automata, Languages, and Programming: 39th International Colloquium, ICALP* 2012.

[12] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *Advances in neural information processing systems*, 2017.

[13] J. Kunegis, "KONECT – The Koblenz Network Collection," in *Proc. Int. Conf. on World Wide Web Companion*, 2013.

[14] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?" in *19th Int. Conf. on WWW*, 2010.

[15] M. Latapy, "Main-memory triangle computations for very large (sparse (power-law)) graphs," *Theoretical computer science*, 2008.

[16] D. Lee, K. Shin, and C. Faloutsos, "Temporal locality-aware sampling for accurate triangle counting in real graph streams," *The VLDB Journal*, vol. 29, no. 6, pp. 1501–1525, 2020.

[17] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, 2014.

[18] Y. Lim and U. Kang, "Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams," in *21st ACM SIGKDD international conference on knowledge discovery and data mining*, 2015.

[19] M. Manjunath, K. Mehlhorn, K. Panagiotou, and H. Sun, "Approximate counting of cycles in streams," in *Algorithms–ESA 2011: 19th Annual European Symposium*, 2011.

[20] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science*, 2002.

[21] M. Mitzenmacher, "A model for learned bloom filters and optimizing by sandwiching," *Adv. in Neural Information Processing Systems*, 2018.

[22] M. Mitzenmacher and S. Vassilvitskii, "Algorithms with predictions," *Communications of the ACM*, 2022.

[23] R. Pagh and C. E. Tsourakakis, "Colorful triangle counting and a mapreduce implementation," *Information Processing Letters*, 2012.

[24] H.-M. Park and C.-W. Chung, "An efficient mapreduce algorithm for counting triangles in a very large graph," in *22nd ACM international conference on Information & Knowledge Management*, 2013.

[25] H.-M. Park, S.-H. Myaeng, and U. Kang, "Pte: Enumerating trillion triangles on distributed systems," in *22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2016.

[26] H.-M. Park, F. Silvestri, U. Kang, and R. Pagh, "Mapreduce triangle enumeration with guarantees," in *23rd ACM International Conference on Information and Knowledge Management*, 2014.

[27] A. Pavan, K. Tangwongsan, S. Tirthapura, and K.-L. Wu, "Counting and sampling triangles from a graph stream," *VLDB Endowment*, 2013.

[28] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015.

[29] K. Shin, "Wrs: Waiting room sampling for accurate triangle counting in real graph streams," in *ICDM*, 2017.

[30] K. Shin, J. Kim, B. Hooi, and C. Faloutsos, "Think before you discard: Accurate triangle counting in graph streams with deletions," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2018.

[31] K. Shin, S. Oh, J. Kim, B. Hooi, and C. Faloutsos, "Fast, accurate and provable triangle counting in fully dynamic graph streams," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2020.

[32] L. D. Stefani, A. Epasto, M. Riondato, and E. Upfal, "Triest: Counting local and global triangles in fully dynamic streams with fixed memory size," *ACM Trans. on Knowledge Discovery from Data (TKDD)*, 2017.

[33] C. Spearman, "The proof and measurement of association between two things." 1961.

[34] S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," in *20th international conference on World wide web*, 2011.

[35] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "Doulion: counting triangles in massive graphs with a coin," in *15th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, 2009.

[36] C. E. Tsourakakis, M. N. Kolountzakis, and G. L. Miller, "Triangle sparsifiers." *J. Graph Algorithms Appl.*, 2011.

[37] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software (TOMS)*, 1985.

Appendix A
Notation and Symbols

Table II resumes the notation and symbols used.

TABLE II
TABLE OF NOTATION AND SYMBOLS USED IN OUR WORK

Symbol	Definition
Notations for Graph Streams	
$G^{(t)} = (V,\ E^{(t)})$	Graph G at time t
$\{u, v\}$	(Unordered) edge between u and v
$\{u, v, +\}$	Addition of edge $\{u, v\}$
$\{u, v, -\}$	Deletion of edge $\{u, v\}$
$\{u, v, w\}$	Triangle with nodes u, v and w
$deg(u)$	Degree of node u
$e^{(t)}$	Edge on the stream at time t
$\Sigma = \{e^{(1)}, ..., e^{(m)}\}$	Arbitrary order graph stream
$\mathcal{T}^{(t)}$	Set of global triangles in $G^{(t)}$
$T^{(t)}$	Number of global triangles in $G^{(t)}$: $\lvert\mathcal{T}^{(t)}\rvert = T^{(t)}$
$\mathcal{T}_u^{(t)}$	Set of local triangles in $G^{(t)}$ for node $u \in V$
$T_u^{(t)}$	Number of local triangles in $G^{(t)}$ for node $u \in V$: $\lvert\mathcal{T}_u^{(t)}\rvert = T_u^{(t)}$
Notations for Algorithms and Analysis	
k	Memory budget, i.e., maximum number of edges that can be stored
$\mathcal{W}$	Waiting room
α	Fraction of the size of $\mathcal{W}$, i.e., $\lvert\mathcal{W}\rvert/k = \alpha, \alpha \in (0, 1)$
$\mathcal{H}$	Set of heavy edges
β	Fraction of the size of $\mathcal{H}$, i.e., $\lvert\mathcal{H}\rvert/k(1-\alpha) = \beta$, $\beta \in (0, 1)$
$\mathcal{S}_L$	Sample set of light edges of size $\lvert\mathcal{S}_L\rvert = s_\ell = k(1-\alpha)(1-\beta)$
$\mathcal{S} = (\hat{V}, \hat{E})$	Subgraph of stored edges: $\mathcal{S} = \mathcal{W} \cup \mathcal{H} \cup \mathcal{S}_L$ of size $\lvert\mathcal{S}\rvert = k$
O_H	Predictor for the measure for heaviness of edges
L	Set of light edges in the stream, of size $\ell = \lvert L\rvert$
$\hat{\mathcal{N}}_u$	Set of neighbors of node u in $\mathcal{S}$
$\hat{T}$	Estimate of global triangles count
$\hat{T}_u$	Estimate of local triangles count for node $u \in \hat{V}$
p_{uvw}	Probability of counting triangle $\{u, v, w\}$

Appendix B
Description of Datasets

In this section, we provide a complete description of the datasets considered in Table I. Recall that, from each dataset, we removed self-loops and multiple edges, deriving a stream of insertion-only edges, for consistency with previous works. We report links for downloading each dataset in https://anonymous.4open.science/r/Tonic-F2C4.

Single graphs:

- *Edit EN Wikibooks* contains the edit network of the English Wikipedia, containing users and pages connected by edit events. This dataset is also considered in [6];
- *SOC Youtube Growth* includes a list of all of the user-to-user links in Youtube video-sharing social network;
- *Cit US Patents* [8] represents the citation graph between US patents, where each edge $\{u, v\}$ indicates that patent u cited patent v (used also in [31]);
- *Actors Collaborations* contains actors connected by an edge if they both appeared in a same movie. Thus, each edge is one collaboration between actors;
- *Stackoverflow* represents interactions from the StackExchange site ”Stackoverflow”. The network is between users, and edges represent three types of interactions: answering a question of another user, commenting on another user's question, and commenting on another user's answer;
- *SOC LiveJournal* is a friendship network from LiveJournal free on-line community.

Snapshot sequences:

- *Oregon* is a sequence of 9 graphs of Autonomous Systems (AS) peering information inferred from Oregon route-views between March 31 2001 and May 26 2001;
- *AS-CAIDA* contains 122 RouteViews BGP graph snapshots, from January 2004 to November 2007;
- *AS-733* are 733 daily instances which span an interval of 785 days from November 8 1997 to January 2 2000, from the BGP logs.
- *Twitter* [4], [14] comprises 4 graphs of the Twitter following/followers network. In all our experiments, we consider each of the 4 single networks independently, and also the larger network *Twitter-merged* obtained by merging the 4 graphs, used also in [32].

Appendix C
Extension to Fully Dynamic Streams

In this section, we describe the adaptation of our algorithm to fully dynamic (FD) graph streams. First, in Section C-A we provide some preliminary notions of random pairing technique, used to maintain size-bounded uniform sample of light edges during the FD stream. Then, in Section C-B we provide a description of our algorithm `Tonic-FD` leveraging random pairing to adapt to FD scenarios.

A. Random Pairing

In our algorithm `Tonic-FD` we sample edges through *random pairing* [7], a method for incrementally maintaining a bounded-size uniform random sample of the items in a dataset in the presence of an arbitrary sequence of insertions and deletions, also adopted in [16], [30]–[32]. The goal of random pairing (RP) is to compensate sample deletions using subsequent insertions: if, at any point, all the previous edge deletions have been compensated, we will have maximum size for our sample, that is $|\mathcal{S}_L| = k(1-\alpha)(1-\beta) = s_\ell$. At any time t in the stream, `Tonic-FD` stores in $\mathcal{S}_L$ a set of $k' \leq s_\ell$ edges sampled uniformly at random among all the *light edges* of $G^{(t)}$, that are edges of $G^{(t)}$ that are neither in the waiting room $\mathcal{W}^{(t)}$ nor kept in the set $\mathcal{H}^{(t)}$ of (predicted) heavy edges. Following RP scheme, `Tonic-FD` maintains counters d_g and d_b for respectively the number of good and the number of bad “uncompensated” deletions. RP works as follows: when receiving an edge deletions, it removes the edge from the sample $\mathcal{S}_L$ if present, and increments the counter d_b for bad deletions, or otherwise it ignores the deletions and just increment the counter d_g for good deletions. When receiving an edge insertion, if the number of uncompensated deletions $d = d_g + d_b = 0$, it proceeds as in standard Reservoir Sampling (see Section III-C). If $d > 0$, we need to account for the uncompensated past deletion(s): we flip a coin and include the incoming edge insertion with probability $d_b/(d_b + d_g)$, or otherwise we exclude the edge insertion from the sample; accordingly, we decrease counter d_b if the edge is added to the sample to compensate for an old bad deletion, or we decrease the counter d_g if the edge is excluded from the sample to compensate for an old good deletion. Let $\mathcal{S}_L^{(t)}$ be the set of edges in $\mathcal{S}_L$ at the end of time step t. Random Pairing guarantees [7] that for every time step t with $|L^{(t)}| \geq k'$, if we let A and B be any two subsets of $L^{(t)}$ of size $|A| = |B| = k'$, then $\mathbb{P}\left[\mathcal{S}_L^{(t)} = A\right] = \mathbb{P}\left[\mathcal{S}_L^{(t)} = B\right]$, that is, any two samples of the same size k' are equally likely to be produced, so that RP is a uniform sampling scheme.

B. Tonic-FD: Counting Triangles with Predictions in Fully Dynamic Streams

In Tonic-FD (Alg. 4), we provide the pseudocode of the adaptation of our algorithm to fully dynamic graph streams. The general workflow of Tonic-FD is very similar to the insertion-only version presented in Section IV. Since we are dealing with both edge insertion and deletions, we make use of random pairing [7], as described in Section C-A.

Our algorithm Tonic-FD works as follows: first, initializes the sets and the counters (lines 1, 2), including d_g and d_b that account for the uncompensated deletions. Then, for each edge in the stream (line 3), given the current edge $\{u, v, \eta\}$, with $\eta \in \{+, -\}$, first it checks whether the edge is an edge insertion ($\eta = +$) or an edge deletion ($\eta = -$). If we have an edge insertion (line 6), we proceed similarly to Alg. 1, but paying attention to the number d_g and d_b of respectively good and bad deletion in the stream observed so far. As described in Section C-A, the number d_g represents the number of uncompensated edge deletions that involved the removal of edges that were not stored in the sample $\mathcal{S}_L$ (i.e., good deletions), while the number d_b is the number of uncompensated deletions of edges that were currently stored in $\mathcal{S}_L$ (i.e., bad deletions), causing the decrement of the size $|\mathcal{S}_L|$ of our stored sample. If such numbers are compensated (i.e., their sum is equal to zero, line 19), Tonic-FD resumes to standard reservoir sampling (see Sect. III-C). Otherwise, we need to account for such uncompensated number of deletions, and sample $\{u', v'\}$ with probability $d_b/(d_b + d_g)$ (line 24). Based on such probability, we compensate the number of deletions by decrementing the respective counter (lines 26, 27).

If the current edge in the stream is an edge deletion (line 28), Tonic-FD checks if the edge to remove is currently in the waiting room $\mathcal{W}$ or in the heavy edges set $\mathcal{H}$ (line 29); in such case, the algorithm removes the edge from the respective set. Otherwise, we are dealing with a removal of a light edge. We first need to decrease the number ℓ of light edges seen in the stream so far (line 31), due to the incoming deletion; then, we have the following cases: (i) the current edge is in the current sample $\mathcal{S}_L$ of light edges. Algorithm 4 removes such edge from $\mathcal{S}_L$ (line 33) and accounts for a bad deletion incrementing d_b counter (line 34). (ii) the current edge is not stored in our sample $\mathcal{S}_L$. Tonic-FD ignores the edge, accounting for a good deletion incrementing d_g counter (line 36).

When returning the final estimates (line 37), we return always non-negative global and local triangle counts: in this way, we trade for unbiasedness of the algorithm (in case of negative estimates) by achieving better accuracy of approximations.

Algorithm 4: Tonic-FD $(\Sigma, k, \alpha, \beta, O_H)$

```
Input  : Arbitrary order fully dynamic edge stream
         $\Sigma = \{e^{(1)}, e^{(2)}, ...\}$; memory budget $k$; fraction of
         waiting room space $\alpha$; fraction of heavy edges
         space $\beta$; edge heaviness predictor $O_H$
Output : Estimate of global triangles count $\hat{T}$; estimate of
         local triangles count $\hat{T}_u$ for each node $u$
1  $\mathcal{W} \longleftarrow \emptyset$; $\mathcal{H} \longleftarrow \emptyset$; $\mathcal{S}_L \longleftarrow \emptyset$;
2  $\hat{T} \longleftarrow 0$; $\ell \longleftarrow 0$; $d_g \longleftarrow 0$; $d_b \longleftarrow 0$;
3  for each edge $e^{(t)} = \{u, v, \eta\}$ in the stream $\Sigma$ do
4      $\mathcal{S} \longleftarrow \mathcal{W} \cup \mathcal{H} \cup \mathcal{S}_L$;
5      CountTriangles-FD($\{u, v, \eta\}, \mathcal{S}, \ell, d_g, d_b$);
6      if $\eta == +$ then
7          if $|\mathcal{W}| < k\alpha$ then $\mathcal{W} \longleftarrow \mathcal{W} \cup \{\{u, v\}\}$;
8          else
9              $\{x, y\} \longleftarrow$ oldest edge in $\mathcal{W}$;
10             $\mathcal{W} \longleftarrow \mathcal{W} \setminus \{\{x, y\}\} \cup \{\{u, v\}\}$;
11             if $|\mathcal{H}| < k(1-\alpha)\beta$ then
12                 $\mathcal{H} \longleftarrow \mathcal{H} \cup \{\{x, y\}\}$;
13             else
14                 $\ell \longleftarrow \ell + 1$;
15                 $\{u', v'\} \longleftarrow$ lightest edge in $\mathcal{H}$;
16                 if $O_H(\{x, y\}) > O_H(\{u', v'\})$ then
17                     $\mathcal{H} \longleftarrow \mathcal{H} \cup \{\{x, y\}\} \setminus \{\{u', v'\}\}$;
18                 else $\{u', v'\} \longleftarrow \{x, y\}$;
19                 if $d_g + d_b == 0$ then
20                     if $|\mathcal{S}_L| < k(1-\alpha)(1-\beta)$ then
21                         $\mathcal{S}_L \longleftarrow \mathcal{S}_L \cup \{\{u', v'\}\}$;
22                     else
23                         SampleLightEdge($\{u, v\}, \mathcal{S}_L, \ell$);
24                 else if FlipBiasedCoin$\left(\frac{d_b}{d_b + d_g}\right)$ == HEAD
                    then
25                     $\mathcal{S}_L \longleftarrow \mathcal{S}_L \cup \{\{u', v'\}\}$;
26                     $d_b \longleftarrow d_b - 1$;
27                 else $d_g \longleftarrow d_g - 1$;
28     else if $\eta == -$ then
29         if $\{u, v\} \in \mathcal{W} \cup \mathcal{H}$ then
             $\mathcal{W} \cup \mathcal{H} \longleftarrow \mathcal{W} \cup \mathcal{H} \setminus \{\{u, v\}\}$;
30         else
31             $\ell \longleftarrow \ell - 1$;
32             if $\{u, v\} \in \mathcal{S}_L$ then
33                 $\mathcal{S}_L \longleftarrow \mathcal{S}_L \setminus \{\{u, v\}\}$;
34                 $d_b \longleftarrow d_b + 1$;
35             else
36                 $d_g \longleftarrow d_g + 1$;
37 return max(0, $\hat{T}$), max(0, $\hat{T}_u$) for each node $u \in \mathcal{S}$;
```

In CountTriangles-FD (Alg. 5), we report the procedure for counting triangles in the current subgraph in the fully dynamic setting. Such algorithm is slightly different from CountTriangles (Alg. 2) since it needs to account for the counters d_g and d_b of good and bad deletions respectively for computing the correction probability, and the type η (i.e., insertion or deletion) of the current edge in order to increment or decrement the global and local triangle estimates.

Appendix D
Proofs

In this section, we provide theoretical proofs of our claims. We describe the correctness (probabilities computation), unbiasedness, variance, time and space complexity of our algo-

Algorithm 5: `CountTriangles-FD`$(\{u,v,\eta\},\mathcal{S},\ell)$

```
Input : edge $\{u,v,\eta\}$; subgraph $\mathcal{S}=\left(\hat{V},\ \hat{E}\right)$; number $\ell$
        of predicted light edges in the stream $\Sigma^{(t)}$; counter
        $d_g$ for uncompensated good deletions; counter $d_b$
        for uncompensated bad deletions
1  for each node $w$ in $\hat{\mathcal{N}}_u \cap \hat{\mathcal{N}}_v$ do
2      initialize $\hat{T}_u$, $\hat{T}_v$, $\hat{T}_w$ to zero if not set yet;
3      $p_{uvw}=1$;
4      if $\{w,u\}\in\mathcal{S}_L$ AND $\{v,w\}\in\mathcal{S}_L$ then
5          $p_{uvw}=\min\left(1,\ \frac{k(1-\alpha)(1-\beta)}{\ell+d_g+d_b}\times\frac{k(1-\alpha)(1-\beta)-1}{\ell+d_g+d_b-1}\right)$;
6      else if $\{w,u\}\in\mathcal{S}_L$ OR $\{v,w\}\in\mathcal{S}_L$ then
7          $p_{uvw}=\min\left(1,\ \frac{k(1-\alpha)(1-\beta)}{\ell+d_g+d_b}\right)$;
8      if $\eta == +$ then
9          increment $\hat{T}$, $\hat{T}_u$, $\hat{T}_v$, $\hat{T}_w$ by $1/p_{uvw}$;
10     else if $\eta == -$ then
11         decrement $\hat{T}$, $\hat{T}_u$, $\hat{T}_v$, $\hat{T}_w$ by $1/p_{uvw}$;
```

rithms `Tonic` and `Tonic-FD`. Also, we prove the propositions of Section IV-A3 on the variance comparison with `WRS` algorithm.

A. Probabilities Computation

In the following, we prove that the probability p_{uvw} computed and used by procedure `CountTriangles`, Alg. 2 (resp. `CountTriangles-FD`, Alg. 5) within `Tonic` (`Tonic-FD`), corresponds to the probability that triangle $\{u,v,w\}$ is counted by `Tonic` (counted or deleted by `Tonic-FD`).

In general, given a triangle $\{u,v,w\}$ discovered when edge $\{u,v\}$ is observed in the stream, the corresponding probability p_{uvw} depends on where the other two edges $\{v,w\},\{w,u\}$ are stored by our algorithm, the counter ℓ of predicted light edges seen so far, and on the number d_g of good and the number d_b of bad deletions if $\{u,v\}$ is an edge deletion. Let us denote without loss of generality $e^{(1)}_{uvw}=\{v,w\}$ as the first edge of the triangle arrived in the stream; similarly $e^{(2)}_{uvw}=\{w,u\}$ as the second one, and $e^{(3)}_{uvw}=\{u,v\}$ as the last one, which corresponds to the edge currently on the stream at the moment in which we discover the triangle $\{u,v,w\}$. We have the following results.

Lemma D.1. *Let $\{u,v\}$ be the edge observed at time t in the* insertion-only *stream and let ℓ be the number of light edges seen up to time t in the stream. For any triangle $\{u,v,w\}$ with last edge $\{u,v\}$, the probability p_{uvw} that $\{u,v,w\}$ is discovered by* `Tonic` *is:*

1) if $\{v,w\}$ and $\{w,u\}\in\mathcal{W}^{(t-1)}\cup\mathcal{H}^{(t-1)}$, then $p_{uvw}=1$;
2) if $\{v,w\}$ and $\{w,u\}\in\mathcal{S}_L^{(t-1)}$, then $p_{uvw}=\min\left\{1,\ \frac{k(1-\alpha)(1-\beta)}{\ell}\times\frac{k(1-\alpha)(1-\beta)-1}{\ell-1}\right\}$;
3) if only one between $\{v,w\}$ and $\{w,u\}$ is in $\mathcal{W}^{(t-1)}\cup\mathcal{H}^{(t-1)}$, then $p_{uvw}=\min\left\{1,\ \frac{k(1-\alpha)(1-\beta)}{\ell}\right\}$;

Proof. Observe that when the edge $e^{(t)}=\{u,v\}$ arrives, the triangle $\{u,v,w\}$ is counted if and only if $\{v,w\}$ and $\{w,u\}$ are in $\mathcal{H}^{(t-1)}\cup\mathcal{W}^{(t-1)}\cup\mathcal{S}_L^{(t-1)}=\mathcal{S}^{(t-1)}$. In the following, we proceed with the proof of the computation of the probabilities by cases as in Lemma D.1:

1) both edges in the waiting room or heavy edge set, i.e., $\{w,u\}\in\mathcal{W}^{(t-1)}\cup\mathcal{H}^{(t-1)}$ and $\{v,w\}\in\mathcal{W}^{(t-1)}\cup\mathcal{H}^{(t-1)}$. Since `Tonic` (Alg. 1) always stores edges in the waiting room and in the heavy edge set (lines 5, 8, 10, 15), then the triangle is counted with probability $p^{(t)}_{uvw}=1$;
2) both edges $\{v,w\}$ and $\{w,u\}$ are in the sample of light edges $\mathcal{S}_L^{(t-1)}$: here, we need to distinguish between two cases. If the sample of light edges $\mathcal{S}_L^{(t-1)}$ is not full yet, i.e., if $\ell\le k(1-\alpha)(1-\beta)$, we have sampled both edges deterministically, hence the discovered triangle is counted with probability 1. Otherwise, when $\ell>k(1-\alpha)(1-\beta)$, we have that:
$$\mathbb{P}\left[\{v,w\}\ \in\mathcal{S}_L^{(t-1)}\text{ and }\{w,u\}\ \in\mathcal{S}_L^{(t-1)}\right]=$$
$$=\mathbb{P}\left[\{v,w\}\ \in\mathcal{S}_L^{(t-1)}\right]$$
$$\times\ \mathbb{P}\left[\{w,u\}\ \in\mathcal{S}_L^{(t-1)}\mid\{v,w\}\ \in\mathcal{S}_L^{(t-1)}\right]=$$
$$=\frac{k(1-\alpha)(1-\beta)}{\ell}\times\frac{k(1-\alpha)(1-\beta)-1}{\ell-1}=p^{(t)}_{uvw};$$
Hence, the overall probability can be written as $p^{(t)}_{uvw}=\min\left\{1,\ \frac{k(1-\alpha)(1-\beta)}{\ell}\times\frac{k(1-\alpha)(1-\beta)-1}{\ell-1}\right\}$;
3) if only one between $\{v,w\}$ and $\{w,u\}$ is in $\mathcal{S}_L^{(t-1)}$, similarly as before, we count the triangle with probability 1 if the sample of light edges is not full yet, else (assuming, without loss of generality, that $\{v,w\}$ is the edge observed in $\mathcal{S}_L^{(t-1)}$ at time t) we can write:
$$\mathbb{P}\left[\{v,w\}\ \in\mathcal{S}_L^{(t-1)}\text{ and }\{w,u\}\ \in\mathcal{H}^{(t-1)}\cup\mathcal{W}^{(t-1)}\right]=$$
$$=\mathbb{P}\left[\{v,w\}\ \in\mathcal{S}_L^{(t-1)}\right]=$$
$$=\frac{k(1-\alpha)(1-\beta)}{\ell}=p^{(t)}_{uvw};$$
Thus, we have that $p^{(t)}_{uvw}=\min\left\{1,\ \frac{k(1-\alpha)(1-\beta)}{\ell}\right\}$.

□

Lemma D.2. *Let $\{u,v\}$ be the edge observed at time t in the* fully dynamic *stream and let ℓ be the number of light edges seen up to time t in the stream. For any triangle $\{u,v,w\}$ with last edge $\{u,v\}$, the probability p_{uvw} that $\{u,v,w\}$ is discovered by* `Tonic-FD` *is:*

1) if $\{v,w\}$ and $\{w,u\}\in\mathcal{W}^{(t-1)}\cup\mathcal{H}^{(t-1)}$, then $p_{uvw}=1$;
2) if $\{v,w\}$ and $\{w,u\}\in\mathcal{S}_L^{(t-1)}$, then $p_{uvw}=\min\left\{1,\ \frac{k(1-\alpha)(1-\beta)}{\ell+d_g+d_b}\times\frac{k(1-\alpha)(1-\beta)-1}{\ell+d_g+d_b-1}\right\}$;
3) if only one between $\{v,w\}$ and $\{w,u\}$ is in $\mathcal{W}^{(t-1)}\cup\mathcal{H}^{(t-1)}$, then $p_{uvw}=\min\left\{1,\ \frac{k(1-\alpha)(1-\beta)}{\ell+d_g+d_b}\right\}$;

Proof. Observe that when the edge $e^{(t)}=\{u,v,\eta\}$ arrives, the triangle $\{u,v,w\}$ is counted (if $\eta=+$) or deleted (if $\eta=-$) if and only if $\{v,w\}$ and $\{w,u\}$ are in $\mathcal{H}^{(t-1)}\cup\mathcal{W}^{(t-1)}\cup\mathcal{S}_L^{(t-1)}=\mathcal{S}^{(t-1)}$. We have:

1) both edges in the waiting room or heavy edge set, i.e., $\{w,u\}\in\mathcal{W}^{(t-1)}\cup\mathcal{H}^{(t-1)}$ and

$\{v, w\} \in \mathcal{W}^{(t-1)} \cup \mathcal{H}^{(t-1)}$. Since Tonic-FD (Alg. 4) always stores edges in the waiting room and in the heavy edge set (lines 7, 10, 12, 17), then the triangle is counted or deleted with probability $p_{uvw}^{(t)} = 1$;

2) both edges $\{v, w\}$ and $\{w, u\}$ are in the sample of light edges $\mathcal{S}_L^{(t-1)}$: here, we need to distinguish between two cases. If the sample of light edges $\mathcal{S}_L^{(t-1)}$ is not full yet, i.e., if $\ell \leq |\mathcal{S}_L^{(t)}|$, we have sampled both edges deterministically, hence the discovered triangle is counted or deleted with probability 1. Otherwise, when $\ell > |\mathcal{S}_L^{(t)}|$, we have that:

$$\begin{aligned}&\mathbb{P}\left[\{v, w\} \in \mathcal{S}_L^{(t-1)} \text{ and } \{w, u\} \in \mathcal{S}_L^{(t-1)}\right] = \\ &= \mathbb{P}\left[\{v, w\} \in \mathcal{S}_L^{(t-1)}\right] \\ &\times \mathbb{P}\left[\{w, u\} \in \mathcal{S}_L^{(t-1)} \mid \{v, w\} \in \mathcal{S}_L^{(t-1)}\right] = \\ &= \frac{k(1-\alpha)(1-\beta)}{\ell + d_g + d_b} \times \frac{k(1-\alpha)(1-\beta) - 1}{\ell + d_g + d_b - 1} = p_{uvw}^{(t)};\end{aligned}$$

Hence, the overall probability can be written as $p_{uvw}^{(t)} = \min\left\{1, \frac{k(1-\alpha)(1-\beta)}{\ell+d_g+d_b} \times \frac{k(1-\alpha)(1-\beta)-1}{\ell+d_b+d_g-1}\right\}$;

3) if only one between $\{v, w\}$ and $\{w, u\}$ is in $\mathcal{S}_L^{(t-1)}$, similarly as before, we count or delete the triangle with probability 1 if the sample of light edges is not full yet, else (assuming, without loss of generality, that $\{v, w\}$ is the edge observed in $\mathcal{S}_L^{(t-1)}$ at time t) we can write:

$$\begin{aligned}&\mathbb{P}\left[\{v, w\} \in \mathcal{S}_L^{(t-1)} \text{ and } \{w, u\} \in \mathcal{H}^{(t-1)} \cup \mathcal{W}^{(t-1)}\right] = \\ &= \mathbb{P}\left[\{v, w\} \in \mathcal{S}_L^{(t-1)}\right] = \\ &= \frac{k(1-\alpha)(1-\beta)}{\ell + d_b + d_g} = p_{uvw}^{(t)};\end{aligned}$$

Thus, we have that $p_{uvw}^{(t)} = \min\left\{1, \frac{k(1-\alpha)(1-\beta)}{\ell+d_b+d_g}\right\}$. □

B. Unbiasedness of Tonic

We now prove that Tonic and Tonic-FD report unbiased estimates of the true global and local triangle counts (Thm IV.1). In our proof we assume that $|\Sigma| > 3$.

Theorem IV.1. *Tonic and Tonic-FD return unbiased estimates of the global triangle count and of the local triangle counts for each node, at any time t. That is, if we let $T^{(t)}$ and $T_u^{(t)}$ be respectively the true global count of triangles in the graph and the true local triangle count for node $u \in V$ at time t, we have:*

$$\mathbb{E}\left[\hat{T}^{(t)}\right] = T^{(t)}, \ \forall\, t \geq 0 \tag{1}$$

$$\mathbb{E}\left[\hat{T}_u^{(t)}\right] = T_u^{(t)} \ \forall u \in V, \ \forall\, t \geq 0 \tag{2}$$

Proof. Let $\delta_{uvw}^{(s)}$ be the random variable representing the amount of increase or decrease of the counters due to the discovery or deletion of the triangle $\{u, v, w\}^{(s)}$ at time $s \leq t$, given the current edge $\{u, v, +\}^{(t)}$ for Tonic (insertion-only), and $\{u, v, \eta\}^{(t)}$ for Tonic-FD. Then, the following holds:

$$\delta_{uvw}^{(s)} = \begin{cases} 1/p_{uvw}^{(s)} & \text{if } \eta = + \text{ and } \{v, w\}, \{w, u\} \in \mathcal{S}^{(s-1)} \\ -1/p_{uvw}^{(s)} & \text{if } \eta = - \text{ and } \{v, w\}, \{w, u\} \in \mathcal{S}^{(s-1)} \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Combining the above with Lemma D.1 and D.2, we have $\mathbb{E}\left[\delta_{uvw}^{(s)}\right] = 1/p_{uvw}^{(s)} \times p_{uvw}^{(s)} \ + \ 0 \ \times (1 - p_{uvw}^{(s)}) = 1$ if the triangle $\{u, v, w\}^{(s)}$ is counted, or $\mathbb{E}\left[\delta_{uvw}^{(s)}\right] = -1/p_{uvw}^{(s)} \times p_{uvw}^{(s)} \ + \ 0 \ \times (1 - p_{uvw}^{(s)}) = -1$ if the triangle $\{u, v, w\}^{(s)}$ is deleted. We can express the estimated number of triangles $\hat{T}^{(t)}$ as $\hat{T}^{(t)} = \sum_{\{u,v,w\}^{(s)} \in \mathcal{T}^{(t)}} \delta_{uvw}^{(s)}$, for $s \leq t$. So, we can write:

$$\begin{aligned}\mathbb{E}\left[\hat{T}^{(t)}\right] &= \mathbb{E}\left[\sum_{\{u,v,w\}^{(s)} \in \mathcal{T}^{(t)}} \delta_{uvw}^{(s)}\right] = \\ &= \sum_{\{u,v,w\}^{(s)} \in \mathcal{T}^{(t)}} \mathbb{E}\left[\delta_{uvw}^{(s)}\right] = \left|\mathcal{T}^{(t)}\right| = T^{(t)}\end{aligned}$$

which proves Eq. 1. Similarly, we can derive the estimated local count for each node u as follows:

$$\begin{aligned}\mathbb{E}\left[\hat{T}_u^{(t)}\right] &= \mathbb{E}\left[\sum_{\{u,v,w\}^{(s)} \in \mathcal{T}_u^{(t)}} \delta_{uvw}^{(s)}\right] = \\ &= \sum_{\{u,v,w\}^{(s)} \in \mathcal{T}_u^{(t)}} \mathbb{E}\left[\delta_{uvw}^{(s)}\right] = \left|\mathcal{T}_u^{(t)}\right| = T_u^{(t)}\end{aligned}$$

which proves Eq. 2. □

C. Time Complexity

We now prove the following theorem that establishes the time complexity of Tonic and Tonic-FD.

Theorem IV.2. *Given an input graph stream Σ, and given $\alpha = |\mathcal{W}|/k$, $\beta = |\mathcal{H}|/k(1-\alpha)$, Tonic and Tonic-FD with memory budget k process each edge in the stream Σ in $\mathcal{O}(k + \log(k(1-\alpha)\beta))$ time.*

Proof. For each incoming edge $\{u, v, \eta\}$ in the stream Σ, the most expensive steps are the computation of common neighbors (line 1 of Alg. 2, line 1 of Alg. 5) and the insertion, retrieval or deletion of the lightest edge in $\mathcal{H}$ (line 13 of Alg. 1, line 15 of Alg. 4). Also, for Tonic-FD such cost is paid if the deletion of current edge occurs in the heavy edge set $\mathcal{H}$ (line 29). In general, we are assuming constant time for: operations in the waiting room $\mathcal{W}$ (implemented through a FIFO queue in Tonic, and through an indexed hash set in Tonic-FD), for querying the predictor O_H, for the coin flip and for accessing, adding or removing an element in the set of light edges $\mathcal{S}_L$, implemented through an array with fixed size s_ℓ. Tonic and Tonic-FD take $\mathcal{O}(k)$ to perform the computation of common neighbors. Then, we need to account for the operations on $\mathcal{H}$; assume that such set is implemented through a min-heap priority queue, where insertion and deletion takes $\mathcal{O}(\log(|\mathcal{H}|))$. Thus, Tonic and Tonic-FD process each incoming edge in $\mathcal{O}(k + \log(k(1-\alpha)\beta))$ time. □

D. Space Complexity

For the space complexity, Tonic and Tonic-FD do not exceed the given fixed memory k while storing the edges of the stream. The proof of the related proposition below is trivial.

Proposition 3. *Given an input graph stream Σ, at the end of the stream, Tonic and Tonic-FD store $\mathcal{O}(k)$ edges to compute the global and local estimates of triangles for the entire graph stream.*

Note that the above analysis does not consider the space required to store the local counters $\hat{T}_u$ for nodes u, which requires an additional $\mathcal{O}\left(\hat{V}^{(t)}\right)$ space, at any time t in the stream. Also, note that Tonic and Tonic-FD explicitly represent only counters for nodes that have at least one adjacent triangles (i.e., only counters > 0).

E. Tonic vs WRS: Variance Comparison

In the following section, we prove Prop. 1. We recall that, for the sake of simplicity, in the analysis we consider a simplified version of WRS algorithm and a simplified version of Tonic, considering *insertion-only* graph streams. The simplified version of the WRS samples each light edge (i.e, that leaves the waiting room) independently with probability p; the simplified version of Tonic uses an oracle that predicts an edge leaving the waiting room as heavy or light, keeps (predicted) heavy edges in $\mathcal{H}$, and samples each light edge with probability $p' < p$, where p is the probability that an edge is sampled by WRS. Note that by properly fixing p', according to the memory budget for heavy edges $(k(1-\alpha)\beta)$ the two algorithms have the same expected memory budget. We assume that the waiting room has the same size in both WRS and Tonic.

We recall that we will assume heavy edges appear in $\geq \rho$ triangles, while light edges appear in $< \rho$ triangles, for some $\rho \geq 3$. However, to capture the errors of the predictor, in particular around the threshold ρ, we assume that edges h predicted as heavy by the predictor are involved in $\Delta(h) \geq \rho/c$ triangles, while edges l predicted as light by the predictor are involved in $\Delta(l) < c\rho$ triangles, for some constant $c \geq 1$. Note that for edges e with $\rho/c < \Delta(e) < c\rho$ the oracle can make arbitrarily wrong predictions. We now prove the following result that provides a condition under which the variance of the estimate of global triangle counts reported by Tonic is smaller than the estimate of WRS.

Proposition 1. *Let $\mathbf{Var}[\hat{T}_{\texttt{WRS}}(p)]$ be the variance of the estimate $\hat{T}_{\texttt{WRS}}$ obtained by WRS when edges are sampled independently with probability p, and let $\mathbf{Var}[\hat{T}_{\texttt{Tonic}}(p')]$ be the variance of the estimate $\hat{T}_{\texttt{Tonic}}$ obtained by Tonic when* light *edges are sampled with probability p'. Then $\mathbf{Var}[\hat{T}_{\texttt{WRS}}(p)] \geq \mathbf{Var}[\hat{T}_{\texttt{Tonic}}(p')]$ if*

$$\frac{T^H}{T^L} > 3\frac{(1/p'^2 - 1/p^2) + c\rho(1/p' - 1/p)}{(1/p-1)(3+4\rho/c)}.$$

Proof. Let $\hat{T}_{algo}$ be the number of triangles estimated by *algo*, where *algo* $\in$ {WRS, Tonic}, and $\mathcal{T}^{S_1,S_2}$ the set of triangles that, when the last edge closing a triangle in such set is observed in the stream, the other two edges are found in S_1 and S_2, where $S_1, S_2 \in \{\mathcal{H}, \mathcal{W}, L\}$. Also, let T^{S_1,S_2}, $\hat{T}^{S_1,S_2}$ be respectively the true and estimated number of triangles in $\mathcal{T}^{S_1,S_2}$. Without loss of generality, we assume that $\mathcal{T}^{S_1,S_2} = \mathcal{T}^{S_2,S_1}$. We recall that triangles counted surely, that is with probability 1, have zero variance (these are triangles in $\mathcal{T}^{\mathcal{W},\mathcal{W}}$ for WRS, and in $\mathcal{T}^{\mathcal{W},\mathcal{W}}, \mathcal{T}^{\mathcal{H},\mathcal{H}}, \mathcal{T}^{\mathcal{W},\mathcal{H}}$ for Tonic). For the other triangles, we have:

$$\begin{aligned}
\mathbf{Var}\left[\hat{T}^{S_1,S_2}_{algo}\right] &= \mathbf{Var}\left[\sum_{\{u,v,w\}\in\mathcal{T}^{S_1,S_2}_{algo}} \delta_{\{u,v,w\}}\right] \\
&= \sum_{\{u,v,w\}\in\mathcal{T}^{S_1,S_2}_{algo}} \mathbf{Var}\left[\delta_{\{u,v,w\}}\right] + \mathbf{Cov}\left[\hat{T}^{S_1,S_2}_{algo}, \hat{T}^{S_1,S_2}_{algo}\right] \\
&= \sum_{\{u,v,w\}\in\mathcal{T}^{S_1,S_2}_{algo}} \left(\mathbb{E}\left[\delta^2_{\{u,v,w\}}\right] - \left(\mathbb{E}\left[\delta_{\{u,v,w\}}\right]\right)^2\right) + \\
&\quad + \mathbf{Cov}\left[\hat{T}^{S_1,S_2}_{algo}, \hat{T}^{S_1,S_2}_{algo}\right] \\
&= T^{S_1,S_2}_{algo}\left(1/q^2 - 1\right) + \mathbf{Cov}\left[\hat{T}^{S_1,S_2}_{algo}, \hat{T}^{S_1,S_2}_{algo}\right]
\end{aligned}$$

where $\delta_{\{u,v,w\}}$ is defined as Eq. 3 (but in a fixed probability context), and $q \in \{p, p^2, p', p'^2\}$ depending on the algorithm for which we are computing the variance and on the set of triangles $\mathcal{T}^{S_1,S_2}$ we are considering. Note that, for WRS we have $q = p$ for triangles in $\mathcal{T}^{\mathcal{H},\mathcal{W}} \cup \mathcal{T}^{L,\mathcal{W}}$ and $q = p^2$ for triangles in $\mathcal{T}^{\mathcal{H},\mathcal{H}} \cup \mathcal{T}^{L,L} \cup \mathcal{T}^{\mathcal{H},L}$; for Tonic, $q = p'$ for triangles in $\mathcal{T}^{\mathcal{H},L} \cup \mathcal{T}^{\mathcal{W},L}$ and $q = p'^2$ for triangles in $\mathcal{T}^{L,L}$.

Thus, for WRS we can write:

$$\begin{aligned}
&\mathbf{Var}\left[\hat{T}_{\texttt{WRS}}(p)\right] = \\
&= \mathbf{Var}\left[\hat{T}^{\mathcal{H},\mathcal{W}}_{\texttt{WRS}} + \hat{T}^{L,\mathcal{W}}_{\texttt{WRS}} + \hat{T}^{\mathcal{H},\mathcal{H}}_{\texttt{WRS}} + \hat{T}^{L,L}_{\texttt{WRS}} + \hat{T}^{\mathcal{H},L}_{\texttt{WRS}}\right] \\
&= \left(T^{\mathcal{H},\mathcal{W}} + T^{L,\mathcal{W}}\right)(1/p - 1) + \\
&+ \left(T^{\mathcal{W},L} + T^{L,L} + T^{\mathcal{H},L}\right)(1/p^2 - 1) + \\
&+ \sum_{x,y,z,w\in[\mathcal{W},\mathcal{H},L]} \mathbf{Cov}\left[\hat{T}^{x,y}, \hat{T}^{w,z}\right]
\end{aligned}$$

Similarly, for Tonic we have:

$$\begin{aligned}
\mathbf{Var}\left[\hat{T}_{\texttt{Tonic}}(p')\right] &= \mathbf{Var}\left[\hat{T}^{L,\mathcal{W}}_{\texttt{Tonic}} + \hat{T}^{\mathcal{H},L}_{\texttt{Tonic}} + \hat{T}^{L,L}_{\texttt{Tonic}}\right] \\
&= \left(T^{\mathcal{H},L} + T^{L,\mathcal{W}}\right)(1/p' - 1) + \\
&+ T^{L,L}(1/p'^2 - 1) + \\
&+ \sum_{x,y,z,w\in[\mathcal{W},\mathcal{H},L]} \mathbf{Cov}\left[\hat{T}^{x,y}, \hat{T}^{w,z}\right]
\end{aligned}$$

where the last sum is intended over all combinations of the sets in $[\mathcal{W}, \mathcal{H}, L]$. We can express the covariance terms as:

$$\mathbf{Cov}\left[\hat{T}^{x,y}, \hat{T}^{w,z}\right] = \sum_{\Delta_i \in \mathcal{T}^{x,y}, \Delta_j \in \mathcal{T}^{w,z}} \mathbf{Cov}\left[\delta_i^{x,y}, \delta_j^{w,z}\right]$$
$$= \sum_{\Delta_i \in \mathcal{T}^{x,y}, \Delta_j \in \mathcal{T}^{w,z}} \mathbb{E}\left[\delta_i^{x,y}, \delta_j^{w,z}\right] - \mathbb{E}\left[\delta_i^{x,y}\right]\mathbb{E}\left[\delta_j^{w,z}\right]$$
$$= \sum_{\substack{\Delta_i, \Delta_j \in \\ \mathcal{T}^{x,y} \cap \mathcal{T}^{w,z}}} \frac{1}{p_i}\frac{1}{p_j}\mathbb{P}\left[\Delta_i, \Delta_j \in \hat{T} \text{ and } \Delta_i \cap \Delta_j \text{ sampled}\right] - 1$$
$$= \sum_{\Delta_i, \Delta_j \in \mathcal{T}^{x,y} \cap \mathcal{T}^{w,z}} \frac{1}{p_i}\frac{1}{p_j}\frac{p_i\, p_j}{q} - 1$$
$$= |\mathcal{T}^{x,y} \times \mathcal{T}^{w,z}| \left(\frac{1}{q} - 1\right)$$

where δ_i, δ_j are respectively the random variable corresponding to the increment in count due to triangle Δ_i and Δ_j; $\Delta_i \cap \Delta_j$ corresponds to the shared edge between triangle Δ_i and triangle Δ_j, $q \in \{p, p'\}$ is the sampling probability of the considered algorithm. Note that only triangles having the shared edge that has been sampled (with probability < 1) have impact on the covariance term. Thus, in order to have non null covariance terms, we need to have $\Delta_i \cap \Delta_j$ that has been sampled as light edge by the respective algorithm.

Therefore, the difference between the variance of $\hat{T}_{\texttt{WRS}}(p)$ and the variance of $\hat{T}_{\texttt{Tonic}}(p')$ can be expressed as:

$$\begin{aligned}
&\mathbf{Var}[\hat{T}_{\texttt{WRS}}(p)] - \mathbf{Var}[\hat{T}_{\texttt{Tonic}}(p')] = T^{\mathcal{H},\mathcal{W}}(1/p - 1) \\
&+ T^{\mathcal{H},\mathcal{H}}(1/p^2 - 1) + T^{\mathcal{H},L}(1/p^2 - 1/p') + 2S_{\mathcal{H}}(1/p - 1) \\
&+ T^{L,\mathcal{W}}(1/p - 1/p') + T^{L,L}(1/p^2 - 1/p'^2) + \\
&+ 2\bar{S}_{\mathcal{H}}(1/p - 1/p'). \qquad (4)
\end{aligned}$$

In Eq. 4, we expressed all the possible combinations of the covariance terms with $S_{\mathcal{H}}$ and $\bar{S}_{\mathcal{H}}$ notation. More explicitly, $S_{\mathcal{H}}$ indicates the number of all pairs of triangles (that share an heavy edge) including at least one deterministic triangle for `Tonic` (i.e., having positive covariance in `WRS` and 0 covariance in `Tonic`), while $\bar{S}_{\mathcal{H}}$ is the number of all pairs of triangles (that share a sampled edge) without deterministic triangles in such pair (i..e., having positive covariance both in `WRS` and in `Tonic`). Furthermore, such pairs in covariance terms in Eq. 4 are uniquely counted, hence we need to multiply the cardinality of both sets by 2.

Note that, assuming that $p' \approx p$ and since $p' < p$, the first four terms are ≥ 0, while the last three are ≤ 0. Let $h \in \mathcal{T}^{\mathcal{H},\mathcal{W}}$ denote an heavy edge: note that $|\mathcal{T}^{\mathcal{H},\mathcal{W}}|\ (1/p - 1) = \sum_{h \in \mathcal{T}^{\mathcal{H},\mathcal{W}}}(1/p - 1)$, and similarly for all other terms not related to $S_{\mathcal{H}}$ and to $\bar{S}_{\mathcal{H}}$. Moreover, note that:

$$S_{\mathcal{H}}(1/p - 1) = \sum_{h \in \mathcal{H}} \binom{\Delta(h)}{2}(1/p - 1),$$

and

$$\bar{S}_{\mathcal{H}}(1/p - 1/p') = \sum_{l \in L} \binom{\Delta(l)}{2}(1/p - 1/p').$$

We have

$$\frac{\Delta(e)^2}{3} \leq \binom{\Delta(e)}{2} \leq \frac{\Delta(e)^2}{2}$$

where the first inequality holds for $\Delta(e) \geq 3$, while the second one is always correct (by considering $\binom{r}{2} = 0$ if $r \leq 1$). The first inequality is used for edges e that are predicted as heavy, therefore requiring $\Delta(e) \geq 3$ is reasonable. We have:

$$\begin{aligned}
\mathbf{Var}[\hat{T}_{\texttt{WRS}}(p)] - \mathbf{Var}[\hat{T}_{\texttt{Tonic}}(p')] &\geq \sum_{h \in \mathcal{T}^{\mathcal{H},\mathcal{W}}}(1/p - 1) \\
+\frac{1}{2}\sum_{h \in \mathcal{T}^{\mathcal{H},\mathcal{H}}}(1/p^2 - 1) &+ \sum_{h \in \mathcal{T}^{\mathcal{H},L}}(1/p^2 - 1/p') \\
+2\sum_{h \in \mathcal{H}}\frac{\Delta(h)^2}{3}(1/p - 1) &+ \sum_{l \in \mathcal{T}^{L,\mathcal{W}}}(1/p - 1/p') \\
+\frac{1}{2}\sum_{l \in \mathcal{T}^{L,L}}(1/p^2 - 1/p'^2) &+ 2\sum_{l \in L}\frac{\Delta(l)^2}{2}(1/p - 1/p').
\end{aligned}$$

Let $T^H = \sum_{h \in \mathcal{H}} \Delta(h)$. We have that:

$$\sum_{h \in \mathcal{H}}\frac{\Delta(h)^2}{3}(1/p - 1) = \sum_{h \in \mathcal{H}}\Delta(h)\frac{\Delta(h)}{3}(1/p - 1) \geq$$
$$\frac{1}{3}(1/p - 1)\rho/c\sum_{h \in \mathcal{H}}\Delta(h) = \frac{1}{3}(1/p - 1)\rho/cT^H$$

since $\Delta(h) \geq \rho/c$ by the assumptions on the predictor. Analogously, Let $T^L = \sum_{l \in L}\Delta(l)$. We have:

$$\sum_{l \in L}\frac{\Delta(l)^2}{2}(1/p - 1/p') = \sum_{l \in L}\Delta(l)\frac{\Delta(l)}{2}(1/p - 1/p') \geq$$
$$\frac{1}{2}(1/p - 1/p')c\rho\sum_{l \in L}\Delta(l) = \frac{1}{2}c\rho(1/p - 1/p')T^L.$$

If $p' \approx p$, we have that $1/p - 1 \leq 1/p^2 - 1/p'$, and $1/p'^2 - 1/p^2 \geq 1/p' - 1/p^2$. Therefore, we have that:

$$\begin{aligned}
&\mathbf{Var}[\hat{T}_{\texttt{WRS}}(p)] - \mathbf{Var}[\hat{T}_{\texttt{Tonic}}(p')] \geq \\
&\geq T^H\left(\frac{1}{6}(1/p - 1)(3 + 4\rho/c)\right) + \\
&+ T^L\left(\frac{1}{2}(1/p^2 - 1/p'^2) + c\rho(1/p - 1/p')\right),
\end{aligned}$$

and $\mathbf{Var}[\hat{T}_{\texttt{WRS}}(p)] - \mathbf{Var}[\hat{T}_{\texttt{Tonic}}(p')] > 0$ if $\frac{T^H}{T^L} > 3\ \frac{(1/p'^2 - 1/p^2) + c\rho(1/p' - 1/p)}{(1/p - 1)(3 + 4\rho/c)}$. ☐

The following result shows that if the predictor O_H provides random predictions, then the variance of the estimate from `Tonic` is the same as the variance of the estimate of `WRS`.

Proposition 2. *If the predictor O_H predicts a random set of edges as heavy edges, and* `WRS` *and* `Tonic` *use the same amount of memory, then* $\mathbf{Var}[\hat{T}_{\mathit{WRS}}(p)] = \mathbf{Var}[\hat{T}_{\mathit{Tonic}}(p')]$.

Proof. (Sketch) If the predictor O_H predicts a random set of edges as heavy edges, then for both `WRS` and `Tonic` the set of edges in memory outside the waiting room is a random subset of light edges. If the two algorithms use the same memory, the two sets have the same cardinality, and the result follows. ☐

Appendix E
Additional Experimental Results

A. Further Experimental Evaluation

1) MinDegree Predictor: Relation to Heavy Edges: Fig. 6 shows the number of triangles (i.e., heaviness) vs the minimum degree of each edge, considering the top 1 ‱ edges sorted by heaviness (for Wikibooks datasets, the top 1000 heaviest edges have been considered). The figure shows that there is a clear and linear relation between the minimum degree of edges and their respective heaviness for the heaviest edge in the graph stream, even if the minimum degree is not a perfect predictor of the heaviness.

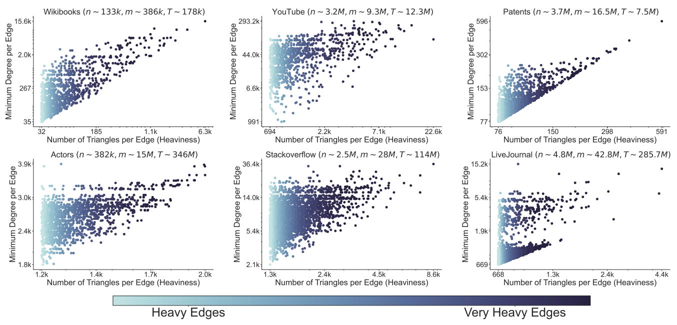

Fig. 6. Relation between Number of Triangles (i.e., heaviness) per edge and Minimum Degree per edge through the top $1‱ m$ heaviest edges in the graph stream (for Wikibooks dataset, the top 1000 heaviest edges have been considered). Both axis are represented using a log scale.

2) Accuracy vs Parameters: the results in Fig. 1 are obtained by running WRS, Chen, and Tonic with values of the main parameters α and β of such algorithms that are:

$$\alpha_{\texttt{WRS}}, \beta_{\texttt{Chen}} \in [0.1, 0.15, 0.19, 0.23, 0.24, 0.28, 0.3, 0.36];$$
$$\alpha_{\texttt{Tonic}}, \beta_{\texttt{Tonic}} \in [0.05, 0.1, 0.15, 0, 2].$$

We chose for such values because we were able to obtain the same space for storing edges with probability 1 for each of the considered algorithm, since the deterministic space accounts for a fraction of the memory budget k corresponding to $\alpha_{\texttt{WRS}}$, $\beta_{\texttt{Chen}}$, and $\alpha_{\texttt{Tonic}} + \beta_{\texttt{Tonic}}(1 - \alpha_{\texttt{Tonic}})$ respectively for WRS, Chen, and Tonic. Since for Tonic some permutation of parameters correspond to the same deterministic space, we report in Fig. 1 the best of such combinations, besides our supposed choice of parameters $\alpha_{\texttt{Tonic}} = 0.05$ and $\beta_{\texttt{Tonic}} = 0.2$ (highlighted by the dashed circle).

3) Quality of Approximation: in the following, we report the probability density function of the estimates of the considered algorithms. In particular, we ran WRS, Chen, Tonic for 500 consecutive and independent trials, and then we estimated the probability density function by using Gaussian Kernel Estimation. As expected, all the algorithms return unbiased estimates, and Tonic is able to achieve a lower variance in all the datasets, leading to more accurate and robust global estimates. The results are shown in figure Fig. 7.

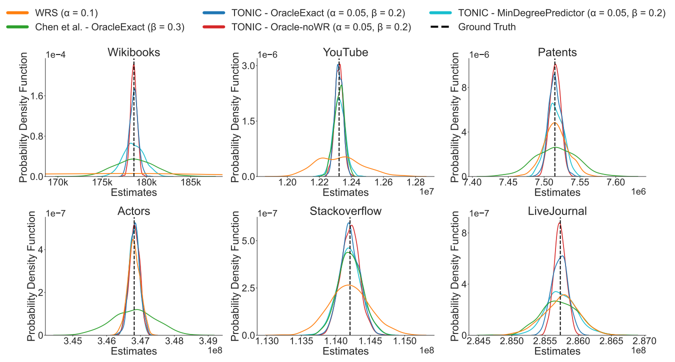

Fig. 7. Probability Density Distribution of estimations. For each combination of algorithm and parameter (including oracle for Tonic), the distribution of estimates over 500 independent repetitions is shown. The algorithms parameters are as in legend (for WRS and Chen they are fixed as in the respective publications; for Tonic they are as chosen in Fig. 1)

4) Accuracy at any time: we studied the behavior of Tonic in the terms of quality of global triangles estimates at any time t, during the evolving of the input graph stream. We considered only WRS since Chen is not suited to return estimates at a given time. Let $t = 1, 2, ..., m$ be the time of arrival of the t-th edge in the input stream Σ. We quantized the time interval in order to obtain around 1k global estimates, hence we collected outputs at times multiple of τ, where $\tau \approx m/1000$. Results are displayed in Fig. 8, where mean and standard deviation are reported. Notice that Tonic is able to significantly outperform WRS for each dataset, for each predictor, and for the all duration of the graph stream, except for the last chunk of stream of Twitter-merged. Also notice that, since here we fixed $k = m/10$ (indicated with an arrow), the error is zero before reaching such number of incoming edges, since we count triangles inside our sample with probability 1, as expected.

5) Number of Discovered Triangles: in Fig. 9, we display the number of counted and estimated triangles for the considered datasets. Notice that Tonic (also the version empowered by MinDegreePredictor) is able, in almost any case, to count more triangles with at least 1 edge in the waiting room and/or in the heavy edge set (i.e., edges that lead to smaller variance), since it takes advantage of features both from the waiting room and from the heavy edges predictions. Also, note that, in general Tonic corrects the number of counted triangles more accurately than Chen, thanks to the reservoir sampling strategy. Again, we recall that the space for storing heavy edges for Chen is $0.3k$, while the space for storing heavy edges in Tonic is $0.19k$.

6) Local Triangles Experiments: we also performed experiments to assess Tonic in estimating local triangle counts. We recall that, by definition, given a node $u \in \mathcal{V}$, the number of local triangles T_u associated to node u is the number of triangles in which u is involved. Note that $\sum_{u \in \mathcal{V}} T_u = 3T$. We considered the top 20% nodes with highest local triangles count, denoted as $V_{top_{20}}$. Given the local triangles estimate $\hat{T}_u$, we computed the following metrics:

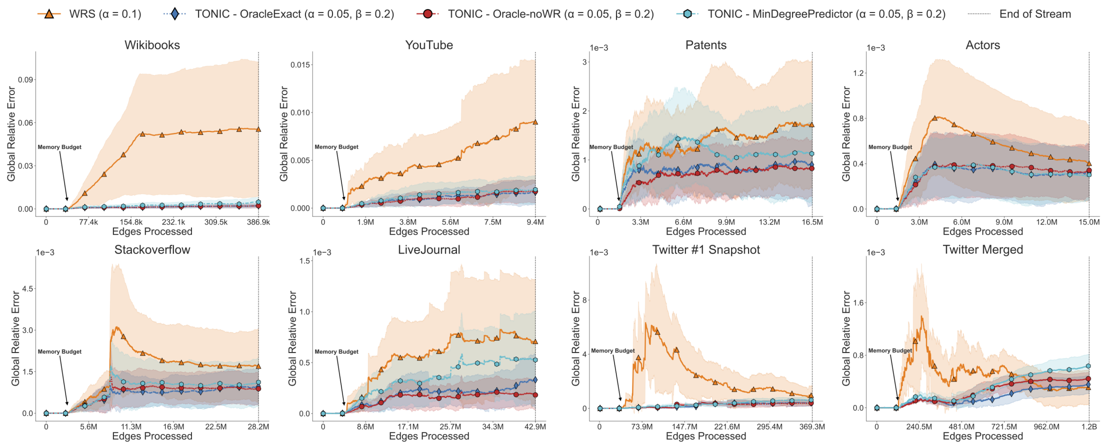
Fig. 8. Estimation error as time progresses during the input graph stream. For each combination of algorithm and parameter (including predictor for Tonic), the average and standard deviation over 50 repetitions are shown (except for Twitter datasets, where 10 repetitions have been considered). The algorithms parameters are as in legend (for WRS they are fixed as in the respective publications; for Tonic they are as chosen in Fig. 1)

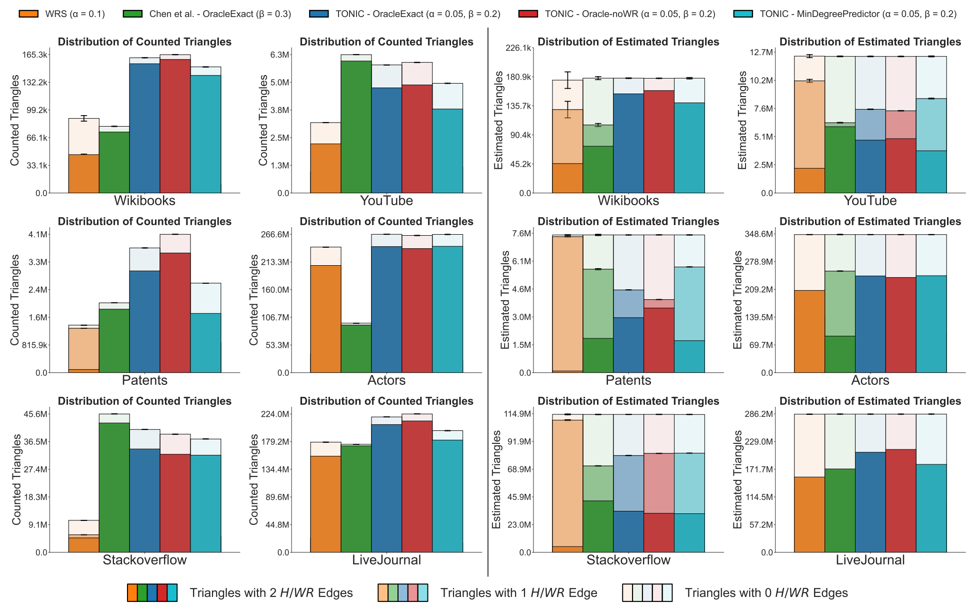
Fig. 9. Distributions of Counted (left) and Estimated (right) Triangles for each dataset. For each combination of algorithm and parameter (including oracle for Tonic), the average and standard deviation over 50 repetitions are shown. The algorithms parameters are as in legend (for WRS and Chen they are fixed as in the respective publications; for Tonic they are as chosen in Fig. 1)

- *Local Relative Error*, defined as:
$$\frac{1}{V_{top_{20}}} \sum_{u \in V_{top_{20}}} \left| \hat{T}_u - T \right| / \, T_u \text{ (the lower the better);}$$
- *Spearman Rank Coefficient* [33] calculated between local triangles ground truth in $V_{top_{20}}$, and the corresponding rank of such nodes in local triangles estimates (the higher the better, max value = 1).

In Fig. 10, we show the results averaged over 10 independent trials, reporting mean and standard deviation. Similarly to Fig. 2, we provided both algorithms with memory budget $k = fm$, with $m = |E|$, for $f = 0.01, 0.03, 0.05, 0.1$. Note that, for some dataset, Tonic is comparable or slightly worse than WRS. This is due to the fact that in Tonic the predictor focuses on heaviest edges, that account for more precise global estimates, at the expense of the estimated count of the local triangles, in particular for vertices with lower local counts.

7) Experiments with Twitter Single Graphs: in this section we provide results also for the second, third and fourth snapshot of Twitter network (first snapshot and the merged version of Twitter have been already discussed in Fig 2). In Fig. 11, we show results for accuracy and runtime by varying the memory budget allowed to the algorithms. Again, Fig. 11 (right) shows the runtime only for Tonic and WRS, since the runtime of Chen is always at least 6 and up to 12 times larger than Tonic runtime. We see that Tonic is always faster than WRS (excluding $1\%m$ memory budget for Twitter #2 snapshot, for which the runtimes are comparable, while

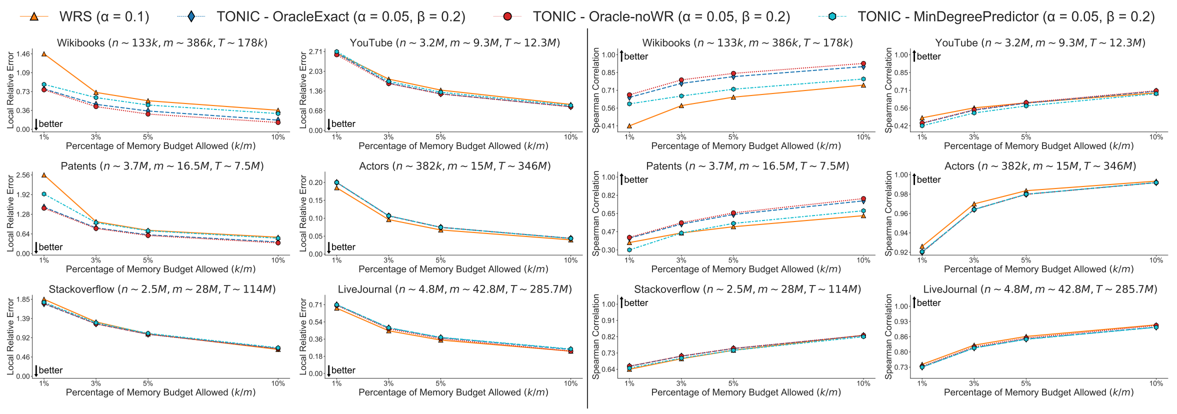
Fig. 10. Error (left) and Spearman Rank Coefficient (right) vs memory budget. For each combination of algorithm and parameter (including predictor for Tonic), the average and standard deviation over 10 repetitions are shown. The algorithms parameters are as in legend (for WRS they are fixed as in the respective publications; for Tonic they are as chosen in Fig. 1)

Tonic is highly outperforming WRS). Such results confirm that Tonic, in practice, is able to scale better with respect to worst-case analyses of the running time. In some cases, Chen has the best accuracy, slightly better than Tonic. This is due to Twitter streams being in adjacency list order, since in such cases the waiting room is not beneficial. In particular, if the current node in the adjacency list stream has a degree comparable to the size of the waiting room, the temporal localities (see Section III-A) are completely lost. However, while the accuracy of Tonic is worse than, but comparable to, Chen, the difference in runtime is huge: for example, in Twitter #2, for a single run with $k = 10\%m$, Tonic takes ~ 15 minutes, Chen requires more than 3 hours.

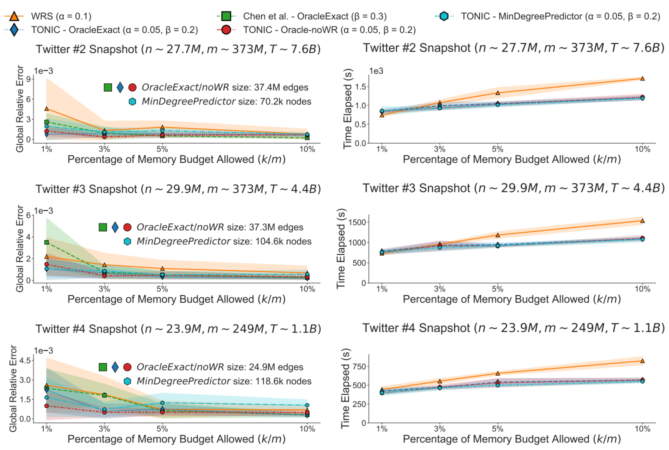
Fig. 11. Error (left) and runtime (right) vs memory budget. Chen runtimes are not shown for clarity, since they are 6-12 times bigger than Tonic runtime. For each combination of algorithm and parameter (including predictor for Tonic), the average and standard deviation over 10 repetitions are shown. The algorithms parameters are as in legend (for WRS they are fixed as in the respective publication; for Tonic they are as chosen in Fig. 1).

8) Adversarial predictors: in this section, we focus on the results of Tonic when it is provided with an extremely bad predictor in snapshot sequences. More specifically, we build *the most adversarial predictor*, that is we set the lightest edges (resp. lowest degree nodes) to be the heaviest (resp. highest degree nodes), and preserving the sizes (i.e., the number of entries in the predictor). In Fig. 12 we report the results containing both best and adversarial predictors. Again, we want to emphasize that our node-degree predictor is taking only some dozens of the node-degrees (the highest degrees in the graph for the best MinDeg predictor, and the lowest ones for the adversarial MinDegPredictor). In the specific, we are only keeping 67 out of 10k, 82 out of 16k nodes, and 34 out of 3k respectively for Oregon, AS-CAIDA and AS-733 training graph, i.e., the first graph in the sequence, as reported in the caption of each subplot. We note that Tonic, even with its adversarial components, is almost always outperforming Chen with the *exact oracle* in AS-CAIDA dataset. In any case, Chen with adversarial oracle is performing absolutely poor with respect to other methods (while Tonic with the adversarial oracle is still competitive). Furthermore, Tonic with adversarial components shows comparable to WRS in AS-CAIDA, AS-733 and for more of the half of the sequence in Oregon. In summary, Fig. 12 shows that Tonic is robust to poor information of adversarial oracles/predictors, thanks to its combination with waiting room sampling, while is able to take advantage of the information provided when they are useful.

9) Fully Dynamic Streams Experiments: in the following, we provide a complete experimental evaluation when dealing with fully dynamic (FD) streams, i.e., with graph streams that allow both insertions and deletions of edges. More specifically, we create FD streams considering our 4 snapshot sequences datasets (see Table I): Oregon, AS-CAIDA, AS-733 and Twitter. For each dataset, starting from the first graph of the sequence, we compute edge additions and removals between the current and the next snapshot. Edge insertions are added to the FD stream by preserving the temporal ordering following the graph sequence, and edge deletions are added to the FD stream with random timestamps inside the time window of the snapshots that we are considering. Hence, at the end of the sequence, we end with our FD stream for each different network of snapshots. In the following, we refer to our algo-

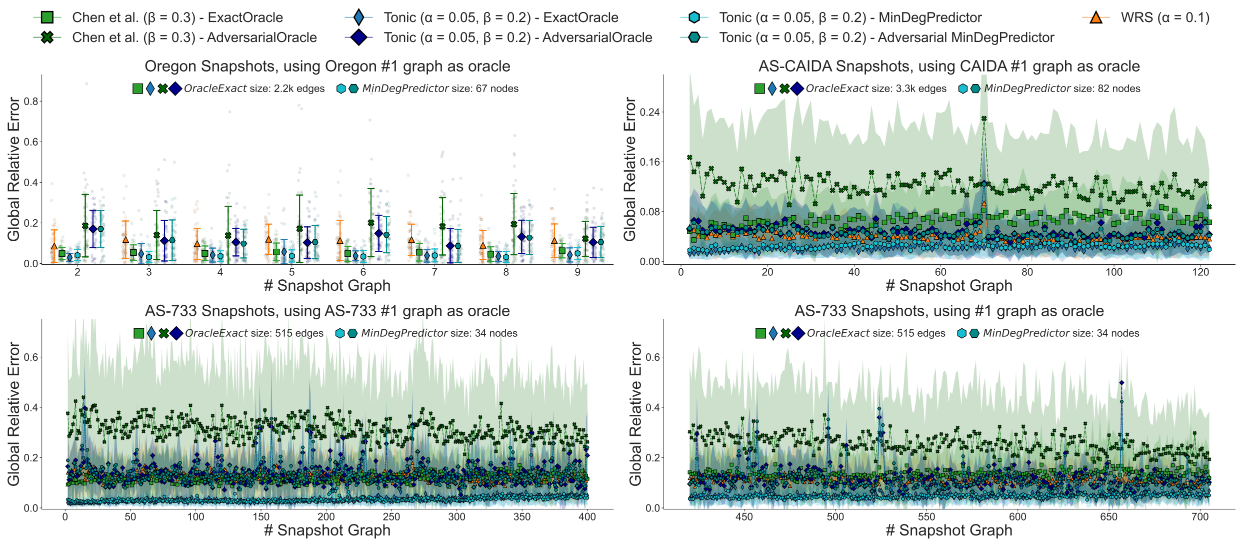

Fig. 12. Error with snapshot networks with sequence of graph streams. In all cases the predictors are trained only on the first graph stream of the sequence (with results not shown on such graph stream). For each combination of algorithm and parameter (including predictor for `Tonic`), the average and standard deviation over 50 repetitions are shown. The algorithms parameters are as in legend (for `WRS` and `Chen` they are fixed as in the respective publications; for `Tonic` they are as chosen in Fig. 1).

rithm for FD streams as `Tonic-FD` for which we described the general workflow in Algorithm 4.

To assess performance of `Tonic-FD` we consider two groups of experiments, similarly to what done for the insertion-only version of our algorithm: (i) error during the evolving of the FD stream, (ii) error and runtime vs memory budget. The oracles and predictors for `Tonic-FD` are trained *only* on the *first* snapshot of the sequence, as described for snapshot experiments in Section V. For each dataset we report, in the title, the following statistics: $\bar{n}$: number of unique nodes; $\bar{m}$: number of unique edges; m_{max}: maximum number of edges at some time; m: total number of edges; T_m: number of global triangles at the end, derived from the FD stream.

Fig. 13 shows the estimation error as time progresses during the input FD stream. Estimates are obtained as in Section E-A4. Notice that `Tonic-FD` performs the best in terms of error in almost any case, for all the considered datasets and all the duration of the FD stream. All the algorithms have been run with memory budget $k = m_{max}/10$ (highlighted by the black arrow). We recall that oracles/predictors for `Tonic-FD` have been trained only on the first graph (corresponding to the insertion-only graph stream Σ_1) and thus completely not aware of the consequent edge deletion.

In Fig. 14 we report results of error and runtimes; the three algorithms are provided with memory budget $k = fm_{max}$, for values of f showed on the x-axis. Note that `Tonic-FD` performs always better than $\texttt{WRS}_{\texttt{del}}$, and better or slightly worse than $\texttt{ThinkD}_{\texttt{acc}}$, in terms of error (Fig. 14, left). Fig. 14 (right) shows that our algorithm `Tonic-FD` is always faster than $\texttt{WRS}_{\texttt{del}}$, and is faster than or comparable to $\texttt{ThinkD}_{\texttt{acc}}$, despite requiring the removal of edges within our waiting room and heavy edge set.

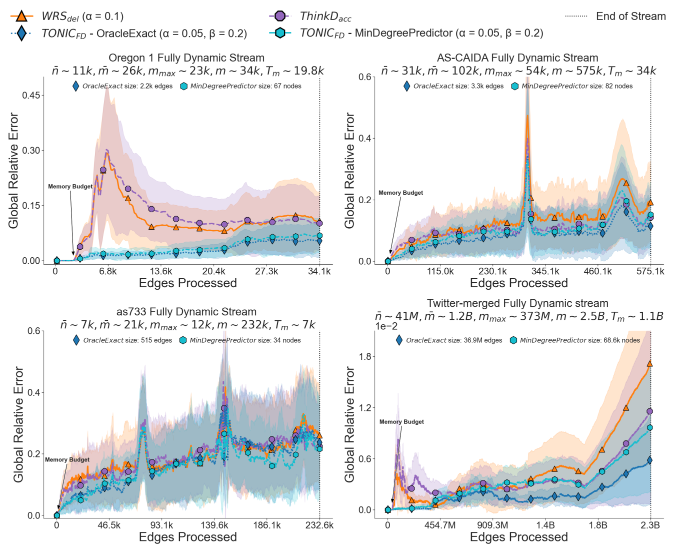

Fig. 13. Estimation error as time progresses during the input FD graph stream. For each combination of algorithm and parameter (including predictor for `Tonic-FD`), the average and standard deviation over 50 repetitions are shown (except for Twitter, where 10 repetitions have been considered). The algorithms parameters are as in legend (for $\texttt{WRS}_{\texttt{del}}$ they are fixed as in the respective publications; for `Tonic-FD` they are as chosen in Fig. 1).

B. Details on Oracles and Predictors

1) Overhead of node-based oracles: in this section, we provide further details on how we build `MinDegPredictor`. More specifically, given the training stream, we sort edges by decreasing minimum degree, and we retain the top 10% of such edges. Thus, each entry of the predictor (represented as a lookup table) can be written as $((u, v); min(deg_u, deg_v))$, for the $m/10$ highest-min degree entries. Starting from

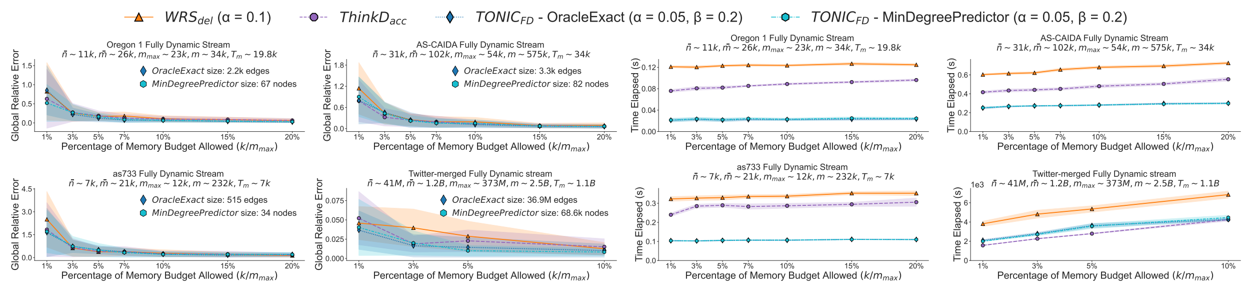
Fig. 14. Error (left) and runtime (right) vs memory budget. For each combination of algorithm and parameter (including predictor for `Tonic-FD`), the average and standard deviation over 50 repetitions are shown (except for Twitter, where 10 repetitions have been considered). The algorithms parameters are as in le end (for $\texttt{WRS}_{\texttt{del}}$ they are fixed as in the respective publications; for `Tonic-FD` they are as chosen in Fig. 1).

such *edge-based* representation, we computed *node-based* `MinDegPredictor`, where the latter contains the highest degree nodes of the whole graph, matching the size of the number of distinct nodes present in the former. Then, in our algorithms, when we query a node-based `MinDegPredictor` for the incoming edge $e = \{u, v\}$, we consider e as *heavy* if both u and v are present in the predictor, or otherwise we consider e as *light*. We want to emphasize the fact that the distinct nodes in the edge-based predictor do not necessarily correspond to the same nodes taken from the highest-degree nodes in the graph: for example, think about a very central and high-degree node that has all low-degrees neighbors, i.e., it would not be among the unique nodes in the edge-based predictor, but it is among the highest-degrees of the graph. Node-based predictors are in someway encoding edge features in a much more succinct representation. Hence, the resulting predictors' overhead is significantly lower than the one used in edge-based representations. Note that in our node-based predictor we combine the degrees to obtain the minimum, but many others predictors based on the degree of nodes could be used (e.g., a linear combination with trainable parameters). We leave the exploration of such predictors as future direction.

TABLE III
STATISTICS ON NODE-BASED `MINDEGREEPREDICTOR` VS EDGE-BASED `MINDEGREEPREDICTOR`. THE FIRST THREE COLUMNS REPRESENT RESPECTIVELY THE RATIO OF THE NUMBER OF ENTRIES, THE SIZE IN MEMORY, AND THE TIME FOR READING FROM FILE OF NODE-BASED PREDICTOR OVER EDGE-BASED PREDICTOR (GAIN FACTORS). THE LAST COLUMNS INDICATES THE JACCARD SIMILARITY.

Dataset	Gain in # of Entries	Gain in Memory	Gain in Time	Jaccard Similarity
Patents	7.90	13.04	9.4 ± 0.3	0.9600
Actors	189.11	288.55	65.0 ± 15.2	1.0
Stackoverflow	147.93	232.72	72.7 ± 16.0	0.9998
Twitter #1 snapshot	538.44	879.57	334.4 ± 188.2	0.9998
Twitter-merged	734.96	1201.52	432.9 ± 284.87	0.9990

In Table III, for each row (dataset) we report the following factors: gain in the number of entries, gain for storing in memory, gain for reading from file, comparing node-based and edge-based representation. We notice that we are able to obtain huge savings in terms of space to store and time to read the predictor. To give additional numbers: for Twitter merged (last row) we have edge-based representation size in memory of $\approx 2.7GB$, and oracle's time to be loaded of $\approx 60s$, while for node-based representation we have respectively $\approx 2.2MB$ and $\approx 0.12s$. Then, we computed the Jaccard Similarity (last column) between the set of unique nodes in edge-based predictor and the set of nodes in node-based predictor, that is computed as the ratio between the cardinality of the intersection and the cardinality of the union of such two sets. Jaccard Similarity provides a measure for gauging similarity and diversity of sets (highest value is 1, corresponding to identical sets). From last column on Table III, we see that nodes contained in the two different representations are basically the same.

2) Time to build predictors/oracles: in the following, we present results for comparison between building `OracleExact` and *node-based* `MinDegPredictor`. We recall that the former requires to solve exactly the problem of counting the number of triangles in the graph, and plus maintaining a lookup table with the value of the heaviness for each edge e, i.e., the number of triangles adjacent to edge e; the latter, is built with a fast pass on the stream and stores the degrees of the nodes in the lookup table. In the end, the edge table (resp. node table) is sorted by heaviness (resp. degree) and the top heaviest edges (resp. highest degrees nodes) are retained. In Table IV we report times for building oracles and predictors for some representative datasets, averaged over 5 independent runs. Note that `Tonic` is able to outperform or is comparable to `WRS` in runtime even if we add times from Table IV to algorithm's execution times in Fig. 2 for most of the combinations of datasets and memory budgets allowed, i.e., if we consider the time for both the first pass for building `MinDegPredictor`, and the second pass to run `Tonic`. This confirms the time practicability and efficiency of our proposed algorithm `Tonic`.

TABLE IV
TIME TO BUILD `ORACLEEXACT` VS TIME TO BUILD `MINDEGREEPREDICTOR`. THE RESULTS INCLUDE ALSO THE TIMES FOR SORTING THE LOOKUP TABLES, RETAINING THE TOP ENTRIES AND WRITING TO FILE. FOR EACH DATASET, AVERAGE AND STANDARD DEVIATION OVER 5 INDEPENDENT REPETITIONS ARE REPORTED.

Dataset	$t_{\texttt{OracleExact}}(s)$	$t_{\texttt{MinDegreePredictor}}(s)$
Patents	44.61 ± 3.15	7.78 ± 0.35
Actors	158.22 ± 7.68	5.68 ± 0.42
Stackoverflow	792.41 ± 28.59	11.27 ± 0.12
Twitter #1 snapshot	$13.35 \times 10^3 \pm 1117.14$	178.47 ± 4.75
Twitter - merged	$64.35 \times 10^3 \pm 6206.57$	561.63 ± 16.42